



**HAL**  
open science

# Codage neural parcimonieux pour un système de vision

Romain Huet

► **To cite this version:**

Romain Huet. Codage neural parcimonieux pour un système de vision. Informatique. Université de Bretagne Sud, 2017. Français. NNT : 2017LORIS439 . tel-01706882

**HAL Id: tel-01706882**

**<https://theses.hal.science/tel-01706882v1>**

Submitted on 12 Feb 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



**THÈSE / UNIVERSITÉ DE BRETAGNE SUD**

**UFR Sciences et Sciences de l'Ingénieur**  
*sous le sceau de l'Université Européenne de Bretagne*

Pour obtenir le grade de :  
**DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE SUD**  
*Mention : Informatique*  
École Doctorale SICMA

présentée par

**Romain Huet**

**IRISA Institut de Recherche en Informatique et  
Systèmes Aléatoires**

# **Codage neuronal parcimonieux pour un système de vision**

**Thèse soutenue le 19-06-2017,**  
devant la commission d'examen composée de :

**Mme. Nicole Vincent**  
Professeur, Université Paris Descartes / Rapporteur

**M. Alain Rakotomamonjy**  
Professeur, Université de Rouen / Rapporteur

**M. Claude Berrou**  
Professeur, Telecom-Bretagne / Examineur

**M. Hubert Cardot**  
Professeur, Université François Rabelais, Tours / Examineur

**M. Philippe Coussy**  
Professeur, Université de Bretagne Sud / Invité

**M. Nicolas Courty**  
Maître de Conférences HDR, Université de Bretagne Sud / Directeur de thèse

**M. Sébastien Lefèvre**  
Professeur, Université de Bretagne Sud / Directeur de thèse



---

## Résumé

Les réseaux de neurones ont connu un vif regain d'intérêt avec le paradigme de l'apprentissage profond ou *deep learning*. Dans le domaine de la vision par ordinateur, ils permettent de surpasser les performances de l'humain sur des tâches bien spécifiques. Cependant, ces réussites s'obtiennent au prix d'importants moyens de calcul à déployer. En effet, même en bénéficiant des dernières avancées des GPU (processeurs embarqués sur cartes graphiques), les besoins en mémoire et en temps de calcul des réseaux profonds restent significatifs. Par conséquent, l'intégration de tels réseaux à des systèmes embarqués (smartphones, puces à faibles ressources) demeure difficile. Alors que les réseaux dits optimisés, de par l'optimisation des paramètres nécessaires pour réaliser un apprentissage, nécessitent de fortes ressources de calcul, nous nous focalisons ici sur des réseaux de neurones dont l'architecture consiste en une mémoire au contenu adressable, appelées mémoires associatives neuronales. Le défi consiste à permettre la réalisation d'opérations traditionnellement obtenues par des calculs en s'appuyant exclusivement sur des mémoires, afin de limiter le besoin en ressources de calcul.

Dans cette thèse, nous étudions une mémoire associative à base de clique, dont le codage neuronal parcimonieux optimise la diversité des données codées dans le réseau. Cette grande diversité permet au réseau à clique d'être plus performant que les autres mémoires associatives dans la récupération des messages stockés en mémoire. Les mémoires associatives sont connues pour leur incapacité à identifier sans ambiguïté les messages qu'elles ont préalablement appris. En effet, en fonction de l'information présente dans le réseau et de son codage, une mémoire peut échouer à retrouver le résultat recherché. Nous nous intéressons à cette problématique et proposons plusieurs contributions afin de réduire les ambiguïtés dans le réseau. Ces réseaux à clique sont en outre incapables de récupérer une information au sein de leurs mémoires si le message à retrouver est inconnu. Nous proposons une réponse à ce problème en introduisant une nouvelle mémoire associative à base de clique qui conserve la capacité correctrice du modèle initial tout en étant capable de hiérarchiser les informations. La hiérarchie s'appuie sur une transformation surjective bidirectionnelle permettant de généraliser une entrée inconnue à l'aide d'une approximation d'informations apprises.

La validation expérimentale des mémoires associatives est le plus souvent réalisée sur des données artificielles de faibles dimensions. Dans le contexte de la vision par ordinateur, nous présentons ici les résultats obtenus avec des jeux de données plus réalistes et représentatifs de la littérature, tels que MNIST, Yale ou CIFAR. Notre modèle, grâce à sa capacité de généralisation non supervisée, atteint des taux de classification allant jusqu'à 70% de succès et ce, en l'espace de moins d'une seconde avec une consommation mémoire minimale. Ces résultats démontrent que l'utilisation des réseaux de neurones est possible sur des supports à ressources limitées.

**Mots clés :** Apprentissage de réseaux de neurones artificiels, Réseaux de neurones à cliques, Mémoire associative, Recherche et généralisation d'informations.

## Abstract

The neural networks have gained a renewed interest through the deep learning paradigm. In the computer vision domain, they allow to surpass human performances on specific tasks. However, these achievements are obtained through the use of massive computational resources. Indeed, even with the most recent graphical processor units (GPU), the needs in memory and computation time for neural nets stay significant. Therefore, their integration in embedded systems (smartphones, low resource chips) remains impossible. While the so called optimised neural nets, by optimizing the parameters necessary for learning, require massive computational resources, we focus here on neural nets designed as addressable content memories, or neural associative memories. The challenge consists in realising operations, traditionally obtained through computation, exclusively with neural memory in order to limit the need in computational resources.

In this thesis, we study an associative memory based on cliques, whose sparse neural coding optimises the data diversity encoded in the network. This large diversity allows the clique based network to be more efficient in messages retrieval from its memory than other neural associative memories. The associative memories are known for their incapacity to identify without ambiguities the messages stored in a saturated memory. Indeed, depending of the information present in the network and its encoding, a memory can fail to retrieve a desired result. We are interested in tackle this issue and propose several contributions in order to reduce the ambiguities in the cliques based neural network. Besides, these cliques based nets are unable to retrieve an information within their memories if the message is unknown. We propose a solution to this problem through a new associative memory based on cliques which preserves the initial network's corrective ability while being able to hierarchise the information. The hierarchy relies on a surjective and bidirectional transition to generalise an unknown input with an approximation of learnt information.

The associative memories' experimental validation is usually based on low dimension artificial dataset. In the computer vision context, we report here the results obtained with real datasets used in the state-of-the-art, such as MNIST, Yale or CIFAR. Our model, thanks to its unsupervised generalisation ability, is able to achieve up to 70% of accuracy in classification, in less than a second with a minimal memory consumption. These results demonstrate that using neural networks is possible on supports with limited resources.

**Keywords :** Machine learning, Artificial neural networks, Cliques based neural networks, Associative memory, Information retrieval and generalisation.

# Table des matières

---

<b>Liste des figures</b>	<b>iii</b>
<b>Liste des tableaux</b>	<b>vi</b>
<b>Liste des algorithmes</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Défis . . . . .	2
1.2 Contributions . . . . .	4
1.3 Plan . . . . .	5
<b>2 État de l’art</b>	<b>7</b>
2.1 Introduction . . . . .	8
2.2 Concepts fondamentaux d’un neurone artificiel . . . . .	8
2.3 Réseaux de neurones . . . . .	10
2.3.1 Réseau feed-forward et rétro-propagation . . . . .	10
2.3.2 Récurrence et mémoire . . . . .	13
2.3.3 Réseaux récurrents à base d’énergie . . . . .	16
2.3.3.1 Machines de Boltzmann . . . . .	17
2.3.3.2 Divergence Contrastive . . . . .	20
2.4 Vers une simplification des calculs . . . . .	21
2.5 Mémoires associatives . . . . .	24
2.5.1 Définition . . . . .	24
2.5.2 Mémoires associatives neuronales (NAM) . . . . .	25
2.5.3 Critères de performance pour un NAM . . . . .	29
2.5.3.1 Efficacité du stockage de l’information . . . . .	29
2.5.3.2 Capacité critique . . . . .	30
2.5.3.3 Avantage de la parcimonie . . . . .	31
2.5.4 Application des NAM . . . . .	32

2.6	Conclusion . . . . .	33
<b>3</b>	<b>Réseau de neurones à cliques</b>	<b>35</b>
3.1	Introduction . . . . .	36
3.2	Modèle de base . . . . .	37
3.2.1	Architecture d'une mémoire . . . . .	37
3.2.2	Phase d'apprentissage . . . . .	38
3.2.3	Phase de récupération . . . . .	42
3.2.3.1	Règles dynamiques . . . . .	42
3.2.3.2	Règles d'activation . . . . .	44
3.3	Analyse des défauts . . . . .	46
3.3.1	Application à des données non uniformes . . . . .	47
3.3.2	Ambiguïtés . . . . .	47
3.3.3	Dimension . . . . .	50
3.3.4	Généralisation . . . . .	52
3.4	Conclusion . . . . .	53
<b>4</b>	<b>Contributions au modèle original</b>	<b>55</b>
4.1	Introduction . . . . .	56
4.2	Gestion de l'ambiguïté . . . . .	57
4.2.1	Solutions de la littérature . . . . .	57
4.2.2	Modèle à couches multiples . . . . .	58
4.2.2.1	Un réseau à double couche . . . . .	58
4.2.2.2	Extension à N couches . . . . .	64
4.2.2.3	Conclusion . . . . .	68
4.2.3	Pénalisation . . . . .	69
4.2.3.1	Expérimentations sur des données artificielles . . . . .	72
4.2.3.2	Expérimentations sur des images . . . . .	74
4.2.3.3	Conclusion . . . . .	75
4.3	Classification . . . . .	76
4.3.1	Méthode . . . . .	78
4.3.2	Expérimentations . . . . .	82
4.4	Généralisation par a priori . . . . .	85
4.4.1	Utilisation d'a priori . . . . .	85
4.4.2	Expérimentations . . . . .	86
4.5	Conclusion . . . . .	89
<b>5</b>	<b>Mémoire associative bidirectionnelle à base de clique</b>	<b>91</b>
5.1	Introduction . . . . .	92
5.2	Concept du modèle BCAM . . . . .	93

---

5.2.1	Phase d'apprentissage . . . . .	96
5.2.2	Phase de récupération . . . . .	97
5.3	Évolution du modèle . . . . .	100
5.3.1	Méthode d'acquisition comprimée . . . . .	100
5.3.2	Topologie adaptative . . . . .	103
5.3.3	Variantes du modèle BCAM . . . . .	107
5.3.3.1	BCAM-BAM . . . . .	107
5.3.3.2	BCAM-CS . . . . .	107
5.4	Expérimentations . . . . .	108
5.4.1	Données générées artificiellement . . . . .	111
5.4.2	MNIST . . . . .	114
5.4.3	Autres jeux d'images . . . . .	118
5.5	Discussion sur la généralisation . . . . .	122
5.5.1	Étude de la dimension des patches . . . . .	126
5.6	Conclusion . . . . .	127
<b>6</b>	<b>Conclusion</b> . . . . .	<b>131</b>
6.1	Contributions . . . . .	132
6.1.1	Ambiguïtés . . . . .	133
6.1.2	Généralisation . . . . .	134
6.2	Perspectives . . . . .	136
	<b>Annexes</b> . . . . .	<b>137</b>
A	Factorisation de matrice . . . . .	139
B	Jeux de données d'images . . . . .	140
B.1	MNIST . . . . .	140
B.2	Yale . . . . .	141
B.3	CIFAR10 . . . . .	142
	<b>Bibliographie</b> . . . . .	<b>143</b>

## Liste des figures

---

1.1	Tendance de recherche du terme <i>Deep Learning</i> . . . . .	3
1.2	Projet SENSE. . . . .	3
1.3	Réseau de neurones à cliques sous forme de mémoire associative. . . . .	3
2.1	Illustration d'un neurone artificiel. . . . .	9
2.2	Illustration de réseaux de neurones artificiels. . . . .	14
2.3	Différence de structure entre deux types de machine de Boltzmann. . . . .	18
2.4	Types de mémoires associatives. . . . .	24
2.5	Architectures de mémoires associatives. . . . .	26
2.6	Capacité critique des NAM en fonction du nombre de neurones dans le réseau. . . . .	31
3.1	Les deux structures possible de CbNN. . . . .	38
3.2	Évolution de la densité. . . . .	40
3.3	Évolution de la densité du réseau CbNN <i>parcimonieux</i> . . . . .	41
3.4	Déroulement de la phase de récupération. . . . .	43
3.5	Taux d'erreur de correction entre des réseaux d'architectures différentes. . . . .	46
3.6	Distribution de l'activité neuronale. . . . .	47
3.7	Répartition dans la matrice d'adjacence. . . . .	48
3.8	Densité et taux d'erreur de correction en fonction du nombre de messages appris. . . . .	49
3.9	Illustration d'une ambiguïté. . . . .	50
3.10	Choix de la quantification. . . . .	51
3.11	Visualisation de la correction par le CbNN d'une image apprise et d'une image inconnue. . . . .	53
4.1	Exemples d'utilisation du modèle à multiples couches de CbNN. . . . .	61
4.2	Évolution des densités des réseaux à deux couches. . . . .	61
4.3	Taux d'erreur de correction des réseaux à deux couches. . . . .	62
4.4	Évolution du temps de calcul pour la phase de récupération. . . . .	63

4.5	Résultats d'expérimentations sur des données artificielles. . . . .	65
4.6	Taux de récupération d'un réseau à couches multiples comparé aux méthodes de la littérature [142]. . . . .	67
4.7	Les effets du paramètre $\beta$ dans la pénalisation. . . . .	71
4.8	Application de la pénalisation sur un exemple d'ambiguïté. . . . .	71
4.9	Effet de la pénalisation sur le taux d'erreurs de correction. . . . .	73
4.10	Impact du paramètre de pénalisation $\beta$ sur la reconstruction d'images. . . . .	76
4.11	Encodage de l'image d'un 3 dans un réseau à clique CbNN. . . . .	79
4.12	Diagrammes des différents types de classification. . . . .	80
4.13	Taux de classification du CbNN et d'un Softmax sur des images non apprises. . . . .	83
4.14	Résultats d'utilisation d'a priori sur des images inconnues au réseau. . . . .	87
4.15	Diagramme de classification pour le CbNN. . . . .	88
5.1	Déroulement de la phase d'apprentissage du BCAM pour un message donné. . . . .	98
5.2	Déroulement de la phase de récupération du BCAM pour un message appris partiellement effacé. . . . .	99
5.3	Déroulement de la phase d'apprentissage du BCAM-CS pour un message donné. . . . .	106
5.4	Déroulement de la phase de récupération du BCAM-CS pour un message appris partiellement effacé. . . . .	106
5.5	Architectures et consommations mémoire des réseaux BCAM sur des données artificielles. . . . .	111
5.6	Résultats de correction sur des données artificielles utilisant des réseaux BCAM. . . . .	112
5.7	Taux de récupération d'un réseau BCAM-BAM $\infty$ comparé aux méthodes de la littérature [142]. . . . .	114
5.8	Architectures et consommations mémoire associées des réseaux BCAM sur les images MNIST. . . . .	115
5.9	Illustration de l'apprentissage d'une image à l'aide du réseau BCAM. . . . .	115
5.10	Résultats de classifications des mémoires associatives à base de clique sur MNIST. . . . .	116
5.11	Exemples de reconstructions à l'aide du BCAM. . . . .	117
5.12	Taux de classification des réseaux à clique comparés à un Softmax sur des images inconnues. . . . .	121
5.13	Indices de performances complémentaires des réseaux à clique. . . . .	122
5.14	Exemples de reconstruction à l'aide des réseaux à cliques. . . . .	123
5.15	Calcul des classements. . . . .	124
5.16	Classements et MSE avec un réseau à forte parcimonie. . . . .	125
5.17	Classements et MSE avec un réseau à faible parcimonie. . . . .	126
5.18	Distributions de l'activation neuronale du BCAM. . . . .	127

---

5.19	Taux de classification en fonction de la dimension des patches. . . . .	128
A.1	Résultats de récupération de l'ensemble des cliques apprises à partir d'une matrice d'adjacence via une factorisation de matrice. . . . .	139
B.1	Visualisation des images du jeu de données MNIST. . . . .	140
B.2	Visualisation de l'ensemble des images du jeu de données Yale. . . . .	141
B.3	Visualisation des images du jeu de données CIFAR10. . . . .	142

# Liste des tableaux

---

2.1	Principales fonctions d'activation de l'état de l'art dans les réseaux de neurones	10
2.2	Capacité critique des mémoires associatives . . . . .	31
4.1	Évaluation des méthodes de la littérature. . . . .	69
4.2	Moyennes des RMSE sur les images MNIST reconstruites par le CbNN. . . . .	76
5.1	Architecture d'un BCAM en fonction d'un découpage des données en $p = 16$ parties. . . . .	95
5.2	Exemples d'architectures avec l'utilisation du Compressive Sensing au sein du BCAM. . . . .	105

# Liste des algorithmes

---

1	Phase d'apprentissage en fonction du type de réseau . . . . .	22
2	Processus de récupération [2] . . . . .	45
3	BCAM : Phase d'apprentissage . . . . .	97
4	BCAM : Phase de récupération . . . . .	99
5	BCAM-CS : Phase d'apprentissage . . . . .	105
6	BCAM-CS : Phase de récupération . . . . .	107
7	Expérimentations avec le BCAM : Vue globale . . . . .	110

# Chapitre 1

## Introduction

---

### Contenu

---

1.1 Défis . . . . .	2
1.2 Contributions . . . . .	4
1.3 Plan . . . . .	5

---

La vision par ordinateur est une discipline relevant de l'intelligence artificielle et s'appuyant sur des tâches d'analyse et de traitement d'images. Elles permettent à un ordinateur de décrire ce qu'il voit [111] en imitant le fonctionnement de la vision humaine. Les tâches comprennent par exemple, la classification pour associer un objet à une classe prédéfinie ; la reconstruction pour compléter, voir recréer une partie d'image manquante ; ou la détection pour identifier des objets spécifiques dans une image.

Les dernières avancées relatives à ces tâches de vision par ordinateur s'appuient sur l'utilisation des réseaux de neurones optimisés (de par l'optimisation des paramètres nécessaires pour réaliser un apprentissage) et profonds (ou *deep neural networks*, DNN). La figure 1.1 illustre l'intérêt porté aux DNN ces dernières années. Une valeur de 100 correspond au pic de popularité pour le terme *deep learning*.

Malgré une démocratisation de l'utilisation de l'outil DNN, illustrée par la figure 1.1, ses performances dans les tâches de vision nécessitent de très larges ressources de calcul reposant sur une exploitation massive des cartes graphiques (GPU). La technologie CUDA de NVIDIA est un langage de programmation facilitant l'exploitation de la puissance des GPU. En effet, la puissance relative par rapport à un processeur central (ou CPU), est décuplée du fait de l'augmentation du nombre d'unités de calcul. Les GPU les plus performants comptent désormais près de 3 000 unités de calcul configurées en parallèle, contre seulement 8 pour les

CPU dont les calculs sont en série. En fonction de l'architecture des réseaux de neurones dits optimisés et des images traitées, l'utilisation du DNN peut nécessiter des heures voire des jours de calculs. Une alternative aux réseaux optimisés visant la simplification des calculs, est de considérer les neurones comme une mémoire au contenu adressable. Ces réseaux de neurones sont appelés mémoires associatives. La différence avec les réseaux optimisés vient de la façon dont les mémoires associatives stockent les informations : directement, sans étape d'optimisation, limitant ainsi les ressources de calcul.

De plus, la plupart des mémoires associatives sont implantées sous forme d'architectures matérielles permettant leur utilisation sur des supports offrant de faibles ressources [86].

Nous pouvons ici rappeler une analogie avec le cerveau humain : ce dernier apprend en temps réel, quand les réseaux optimisés ont besoin d'un temps de calcul pour assimiler l'information. Les mémoires associatives, à l'instar du cerveau, intègrent directement les données. Les réseaux optimisés sont l'équivalent de la mémoire à long terme, tandis que les mémoires associatives correspondent à la mémoire à court terme.

Notre travail de recherche s'inscrit dans ce contexte de simplification des calculs afin d'effectuer des tâches de vision par ordinateur à l'aide de mémoires associatives. Les recherches ont contribué au projet SENSE consistant à recréer le parcours de la vision humaine via l'utilisation d'architectures physiques à base de réseaux de neurones. Le projet fédère trois thématiques de recherche. La première étape de la vision consiste en la transformation électrique des signaux lumineux par la rétine. Le premier axe du projet correspond aux travaux sur le développement de capteurs CMOS pour l'acquisition des signaux lumineux. Ces signaux nouvellement transformés sont ensuite envoyés dans la partie visuelle du cortex via le nerf optique en passant par le noyau géniculé latéral (LGN) situé dans le thalamus, l'aire primaire et les autres parties corticales du cerveau. Les travaux du second axe reposent sur le développement d'une architecture matérielle pour mimiquer le passage des signaux électriques à travers les aires visuelles du cortex. Le dernier axe qui fait l'objet de la présente étude consiste à traiter ces données numériquement pour des tâches de vision par ordinateur. La figure 1.2 présente les diverses missions entreprises dans ce projet collaboratif dont la nôtre est entourée en rouge. Cette dernière est financé à travers une bourse de la région Bretagne en partenariat avec CominLabs faisant partie du programme des laboratoires d'excellence du grand Ouest qui finance le projet SENSE.

## 1.1 Défis

L'utilisation exclusive de mémoires pour effectuer des calculs est le principal défi de ce travail. Cependant, le cadre du projet SENSE apporte des contraintes supplémentaires dont la principale est de concentrer les recherches sur une mémoire associative spécifique, le réseau neuronal de Gripon et Berrou (GBNN) [52]. Ce réseau a été développé dans le cadre des

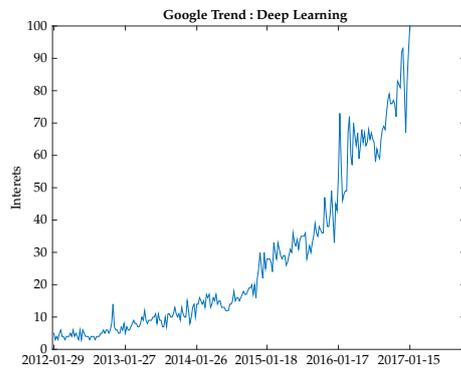


FIGURE 1.1 – *Tendance de recherche du terme Deep Learning.*

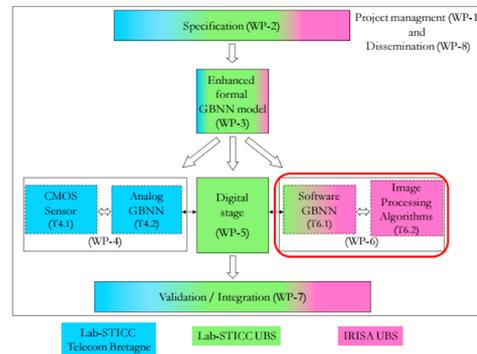


FIGURE 1.2 – *Projet SENSE.*

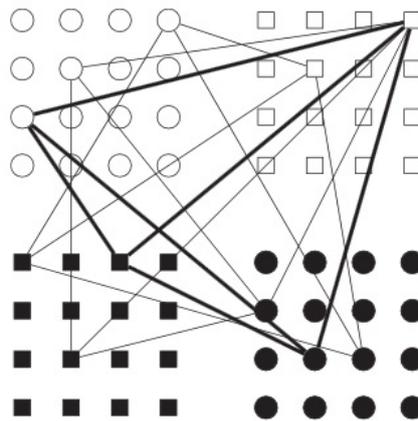


FIGURE 1.3 – *Réseau de neurones à cliques sous forme de mémoire associative. Figure issue de l'article Sparse neural networks with large learning diversity [52].*

codes correcteurs. Les informations apprises sont codées dans la mémoire associative sous forme de graphes complets appelés *cliques*. L'utilisation du réseau repose sur la correction d'informations apprises au préalable par la mémoire via une complétion de graphe. La figure 1.3 illustre le codage des données en cliques. La récupération de ces données s'effectue par inférence à partir des messages appris.

Le choix d'un tel réseau repose sur sa simplicité en terme d'apprentissage, de correction de l'information, et sa portabilité sur une architecture matérielle [73] [74] [75]. Le principal objectif des travaux décrits dans ce manuscrit est d'employer le GBNN pour des tâches de vision par ordinateur tout en minimisant son besoin en ressources afin de pouvoir le déployer sur des puces à faible consommation. La validation du modèle GBNN a été faite sur des messages générés aléatoirement suivant une distribution uniforme. Ainsi l'utilisation d'images par le réseau soulève trois problématiques : la dimension, les ambiguïtés et la généralisation.

- Les données artificielles étaient de dimension faible afin de simplifier l'intégration du réseau sur des puces à faibles consommation de ressources. L'utilisation d'images entraîne une nette augmentation de la dimension du GBNN nécessaire à l'apprentissage et donc des ressources nécessaires. Le défi consiste à réduire cette dimension pour que le réseau puisse toujours être utilisé par des puces à faible consommation.
- Le codage de l'information dans le réseau passe par l'activation des neurones associés pour former une clique. Une fois activés, les neurones le restent à jamais. Ainsi l'utilisation de données dont la distribution est non uniforme, comme des images, entraîne une sur-utilisation des mêmes neurones. La concentration des informations conduit le réseau à ne plus être en mesure de distinguer les cliques apprises, donnant lieu au phénomène d'ambiguïté. Il est alors nécessaire d'aider le réseau à faire la distinction entre les messages qu'il a appris, que ces derniers suivent une distribution uniforme ou non.
- Dans l'esprit des codes correcteurs, le réseau est utilisé pour effectuer la correction sur des données partiellement effacées. Les performances élevées en reconstruction sont liées au fait que les données dégradées ont été préalablement apprises sans dégradation. La difficulté est de permettre au réseau de généraliser les informations apprises pour corriger des données inconnues.

Nos contributions consistent à résoudre ces différentes problématiques.

## 1.2 Contributions

Au vu des modifications effectuées sur le réseau initial GBNN, nous l'appellerons désormais réseau de neurones à base de cliques (ou *Clique based Neural Network*, CbNN). Les contributions décrites dans ce manuscrit s'articulent autour de deux axes majeurs, la gestion des ambiguïtés et la capacité de généralisation du réseau. Nous résolvons en partie la problématique de la dimension des images par l'utilisation d'une quantification. Cette solution n'est pas optimale de par l'apport de biais apporté par cette méthode.

Nous traitons la problématique des ambiguïtés au travers des travaux suivants.

- Nous présentons un modèle composé de deux couches de CbNN pour illustrer l'apport de la plasticité au réseau sur la correction de cliques distinctes pour limiter l'apparition des ambiguïtés. Le réseau est ensuite étendu à  $N$  couches. Notre contribution repose sur l'interaction entre les couches permettant la plasticité.
- Nous proposons une méthode de pénalisation promouvant l'activation des neurones représentant une unique clique pour limiter la production de superposition des graphes et donc des ambiguïtés. La pénalisation passe par une contrainte soumise aux activations des neurones durant la phase de récupération des données de la mé-

moire. Les résultats sont issus d'expérimentations sur des données artificielles et sur des images.

Après avoir proposé des réponses aux problèmes d'ambiguïtés, nous discutons de la notion de généralisation au travers des contributions suivantes.

- Nous proposons une stratégie d'approximation de cliques basée sur la méthode de pénalisation. Le réseau cherche, à partir d'une entrée inconnue, à reconstruire une clique. Mais du fait de la méconnaissance de la clique d'entrée, la phase de récupération produit un chevauchement de cliques. Nous traitons ce problème en ciblant l'activation des neurones proches de l'entrée inconnue soumise, soit une reconstruction à base d'a priori. Notre contribution illustre les limites des mécaniques de récupération liées à l'architecture du CbNN.
- En s'appuyant sur les résultats de la généralisation par a priori, nous proposons une nouvelle architecture de mémoire associative à base de réseau à cliques. Le noyau du nouveau réseau repose sur le passage bidirectionnel de l'information entre plusieurs couches de CbNN, d'où le nom de Mémoire Associative Bidirectionnelle à base de Clique (ou *Bidirectional Cliques based Associative Memory*, BCAM). Notre contribution consiste en le développement d'un réseau à base de cliques capable de généraliser l'information apprise tout en gardant sa capacité correctrice initiale. Le point fort de cette méthode, en plus de sa capacité à généraliser, est la minimisation de l'impact mémoire entraînant un temps de récupération réduit.

Les contributions sur l'ambiguïté et la généralisation sont toutes appliquées en premier lieu à une problématique de reconstruction. Nous proposons également une modification de l'architecture du CbNN pour permettre la classification d'images en utilisant les mécaniques de récupération du réseau.

## 1.3 Plan

Le manuscrit suit le plan suivant. Le prochain chapitre discute de la nécessité de simplifier les réseaux de neurones, et justifie l'utilisation des mémoires associatives. Ce chapitre a pour but d'aider à faire la distinction entre les réseaux dits optimisés et les mémoires associatives. De plus, nous analysons les performances de ces mémoires afin de motiver notre utilisation du GBNN et son évolution appelée CbNN par rapport aux autres mémoires associatives. Le chapitre 3 présente le réseau à clique (CbNN) en détaillant ses mécaniques d'apprentissage et de récupération de l'information. Le chapitre permet de mettre en avant les difficultés liées à l'utilisation du réseau CbNN avec des données non uniformes et inconnues. Dans le chapitre 4 nous comparons nos propositions aux solutions de l'état de l'art

traitant des problématiques d'ambiguïtés et de généralisation. Nous expliquons aussi comment utiliser le CbNN pour des tâches de classification. Nous étendons nos contributions en généralisation en proposant au chapitre 5 la nouvelle architecture BCAM pour la gestion de données inconnues au réseau. Nous finissons le manuscrit par une conclusion générale sur les mémoires associatives et leurs utilisations alternatives aux réseaux optimisés dans une optique de simplification de calculs.

# Chapitre 2

## État de l'art

---

### Contenu

<b>2.1</b>	<b>Introduction</b>	<b>8</b>
<b>2.2</b>	<b>Concepts fondamentaux d'un neurone artificiel</b>	<b>8</b>
<b>2.3</b>	<b>Réseaux de neurones</b>	<b>10</b>
2.3.1	Réseau feed-forward et rétro-propagation	10
2.3.2	Récurrance et mémoire	13
2.3.3	Réseaux récurrents à base d'énergie	16
2.3.3.1	Machines de Boltzmann	17
2.3.3.2	Divergence Contrastive	20
<b>2.4</b>	<b>Vers une simplification des calculs</b>	<b>21</b>
<b>2.5</b>	<b>Mémoires associatives</b>	<b>24</b>
2.5.1	Définition	24
2.5.2	Mémoires associatives neuronales (NAM)	25
2.5.3	Critères de performance pour un NAM	29
2.5.3.1	Efficacité du stockage de l'information	29
2.5.3.2	Capacité critique	30
2.5.3.3	Avantage de la parcimonie	31
2.5.4	Application des NAM	32
<b>2.6</b>	<b>Conclusion</b>	<b>33</b>

---

## 2.1 Introduction

Les ordinateurs actuels reposent sur une architecture de Von Neumann ; même les ordinateurs quantiques ont pour optique d’être développés selon ce type d’architecture [97]. L’architecture de Von Neumann est basée sur l’interaction d’une unité de calcul avec une unité de mémoire pour traiter des données. L’interaction repose sur une compréhension d’antan du cerveau humain [135] en le considérant comme une machine alliant un traitement numérique et analogique de l’information avec des opérateurs logiques. Afin d’imiter l’activité du cerveau humain au travers d’opérateurs logiques, une alternative consiste en l’utilisation des réseaux de neurones.

Nous rappelons dans ce chapitre les concepts d’un neurone artificiel et l’agrégation de neurones au sens d’un réseau permettant désormais de dépasser l’humain dans des tâches spécifiques de vision par ordinateur, comme la classification d’images [57]. La suite du chapitre explique comment utiliser ces neurones. Chaque unité de calcul possède une mémoire ponctuelle. Les réseaux composés de mémoires supplémentaires sont les plus performants [115]. Cet apport rejoint l’idée de l’architecture de Von Neumann. Un dernier type de réseaux de neurones utilise les unités de calcul sous forme de mémoire adressable, ce sont les mémoires associatives neuronales.

L’objectif principal de ce chapitre est de justifier l’utilisation des mémoires associatives (objet de la présente thèse) au lieu des autres types de réseaux de neurones. L’argument principal consiste en la simplification des calculs et, par conséquent, la minimisation des ressources utilisées résultant en une économie d’énergie, une diminution du temps de calcul, etc. Nous décrivons les processus d’apprentissage et de récupération d’informations des différents types de réseaux afin de souligner notre intérêt pour les mémoires associatives. L’étude de ces dernières s’est inscrite dans le contexte du projet SENSE qui se focalise sur les réseaux de neurones à cliques.

Ce chapitre s’articule autour de deux parties. La première partie rappelle le concept d’un neurone artificiel et l’agrégation de neurones artificiels en un réseau. Pour cela, nous rappelons les principes d’apprentissage et de récupération d’informations de différents types de réseaux de neurones, ce qui nous permet de motiver l’utilisation des mémoires associatives. La seconde partie porte sur ces mémoires neuronales en exposant leurs définitions et leurs performances, justifiant ainsi l’utilisation du CbNN dans nos travaux.

## 2.2 Concepts fondamentaux d’un neurone artificiel

Le premier neurone artificiel a été développé en 1943 par McCulloch et Pitts [99]. Il consistait en une unité de calcul effectuant des opérations  $\varphi$  logiques *OU* ou *ET* sur ses

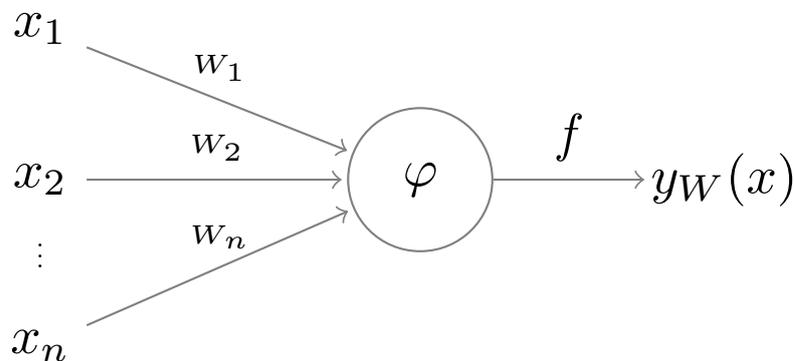


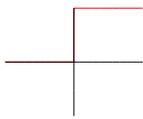
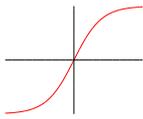
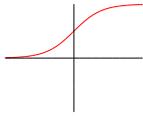
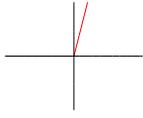
FIGURE 2.1 – Illustration d’un neurone artificiel.

entrées  $x_i$  afin d’obtenir en sortie un résultat  $y$  binaire à travers la fonction d’activation de seuil définie telle que  $y = f(\varphi) = 1$  si  $\varphi \leq \theta$ ,  $y = f(\varphi) = 0$  sinon. La figure 2.1 illustre un neurone artificiel et sa capacité de calcul.

L’inconvénient d’un tel neurone est sa restriction aux données binaires. Une nouvelle forme d’apprentissage via ce type d’unité de calcul se base sur la plasticité synaptique définie par le psychologue Donald Hebb en 1949. Il émettait l’hypothèse, vérifiée par la suite [28], que la force de connexion entre deux neurones évolue en fonction de leurs activités communes [58]. On parle de règle d’apprentissage de type Hebbien quand le réseau de neurones repose sur une matrice  $W$  correspondant aux poids de connexions évoluant au cours du temps.

Le Perceptron de Rosenblatt [112] s’appuie sur les travaux de McCulloch et Pitts pour le neurone et de Hebb pour les poids de connexions. Désormais, les opérations  $\varphi$  ne s’effectuent plus avec des variables binaires mais réelles compte tenu des propriétés d’apprentissage Hebbien. Les variables changent de représentation à l’application d’une fonction d’activation sur la somme de leurs connexions entrantes, soit  $\varphi = \sum_{i=1}^n W_i x_i$ . Bien que la fonction d’activation de seuil s’appliquait pour le premier neurone, on distingue désormais un ensemble grandissant de ces fonctions, dont les plus couramment utilisées sont listées dans le tableau 2.1. Ce tableau liste les fonctions d’activation par leurs noms et indique leurs représentations visuelles, équations associées, et espaces de projections.

Une généralisation du Perceptron consiste à utiliser plusieurs neurones connectés les uns aux autres afin de former un réseau.

Nom	Graphe	Equation	Projection
Seuillage		$f(\varphi) = \begin{cases} 0, & \text{if } \varphi < 0. \\ 1, & \text{sinon.} \end{cases}$	$\{0;1\}$
Tangente Hyperbolique		$f(\varphi) = \frac{2}{1+\exp(-2\varphi)} - 1$	$(-1,1)$
Sigmoïde		$f(\varphi) = \frac{1}{1+\exp(-\varphi)}$	$(0,1)$
Unité Linéaire Rectifiée		$f(\varphi) = \begin{cases} 0, & \text{if } \varphi < 0. \\ x, & \text{sinon.} \end{cases}$	$[0,\infty)$

**TABLEAU 2.1** – Principales fonctions d'activation de l'état de l'art dans les réseaux de neurones. Le ratio abscisse pour ordonnée des graphes est de 5 pour 1.

## 2.3 Réseaux de neurones

La topologie d'un réseau de neurones, *c.-à-d.* son nombre d'unités et le type de connexion, dépend de la tâche pour laquelle il a été conçu. Par exemple, le Perceptron est utilisé dans le cas de classification binaire. Cette tâche nécessite une architecture vers l'avant (ou *feed-forward*, FF) du fait que le passage de l'information à travers le réseau s'effectue de manière unilatérale comme illustré par la figure 2.2a.

### 2.3.1 Réseau *feed-forward* et rétro-propagation

Le Perceptron a évolué en multi-Perceptron pour classer un nombre d'objets supérieur à deux. L'idée générale d'un réseau FF est de transformer les entrées soumises dans un nouvel espace à l'aide d'un changement de dimension pour simplifier les données à classer. En effet, les réseaux de neurones, à travers leurs couches cachées, apprennent des fonctions non-linéaires (à l'aide de la fonction d'activation telle que la sigmoïde) afin de classer linéairement des variables non séparables [93]. Cette nouvelle représentation des données  $x$  est faite par transformation linéaire de la matrice de poids  $W$  qui correspond à la pente de la fonction évaluée par les neurones de la couche cachée. Un biais  $b$  translate la fonction. Puis, la fonction d'activation  $f$  agit ponctuellement sur chacun des neurones de manière à projeter les données dans leurs nouvelles représentations. En reprenant l'illustration d'un réseau FF

de la figure 2.2a, la sortie des neurones de la couche cachée est définie par

$$y_{W,b}(x) = f(Wx + b). \quad (2.1)$$

En présence de plusieurs couches, la succession d'informations nécessitant une réduction de dimension accrue se fait comme suit :

$$\begin{aligned} z_i^{(2)} &= \sum_{j=1}^{n^{(2)}} W_{i,j}^{(1)} a_j^{(1)} + b_i^{(1)}, \\ a_i^{(2)} &= f(z_i^{(2)}), \\ &\vdots \\ z_i^{(l)} &= \sum_{j=1}^{n^{(l)}} W_{i,j}^{(l-1)} a_j^{(l-1)} + b_i^{(l-1)}, \end{aligned} \quad (2.2)$$

$$a_i^{(l)} = f(z_i^{(l)}), \quad (2.3)$$

pour  $2 \leq l \leq ol$  avec  $ol$  le nombre correspondant à la couche de sortie. Nous remarquons que  $a_j^{(1)} = x_j$  avec  $x$  correspondant aux données soumises à l'entrée du réseau. Dans les équations ci-dessus,  $z_i^{(l)}$  correspond aux combinaisons linéaires à l'entrée d'un neurone  $i$  de la couche  $l$ , et  $a_i^{(l)}$  est sa valeur de sortie à travers une fonction d'activation. L'apprentissage d'un tel réseau repose sur l'optimisation des poids  $W$  et biais  $b$  à l'aide de l'algorithme de rétro-propagation (ou *back propagation*, BP) [113]. L'optimisation cherche à minimiser le gradient de la fonction de coût, généralement définie par l'erreur quadratique entre la sortie du réseau  $\hat{y}_{W,b}(x)$  et la sortie désirée  $y$ ,

$$\operatorname{argmin}_{W,b} E(W,b) = \frac{1}{2} \|y - \hat{y}_{W,b}(x)\|^2. \quad (2.4)$$

Par exemple, dans le cas de l'apprentissage supervisé, la sortie du réseau est connue. La minimisation est faite entre le résultat du réseau et ce qui est souhaité (une étiquette pour une tâche de classification). En effet, le passage de l'information à travers les couches entraîne une erreur qui se diffuse proportionnellement au nombre de couches contenues dans le réseau. La rétro-propagation corrige cette erreur en la calculant couche par couche, en commençant par la dernière à travers la comparaison entre le résultat issu du réseau et la référence. Par conséquent, la correction des erreurs entraîne l'évolution des poids et biais correspondant ainsi à l'apprentissage du réseau.

L'évolution des poids s'effectue par itérations d'une descente de gradient. Chaque itération effectue un pas proportionnel vers la direction opposée au gradient. Cette méthode cherche un minimum local d'une fonction, dans notre cas l'équation (2.4), pour un point donné. La convergence à l'aide d'une descente de gradient vers un minimum global est garantie si la fonction de coût suit en particulier les hypothèses suivantes : (1) la fonction est

convexe et finie pour toutes ses valeurs, (2) un minimiseur fini existe, et (3) le gradient de la fonction est Lipschitz continue avec une constante  $L > 0$ . La garantie de la convergence avec les hypothèses ci-dessus est valide si le pas de descente est constant et de préférence petit. Dans le cas d'une classification, la fonction de coût majoritairement utilisée est la fonction d'erreur quadratique qui suit les hypothèses précédentes permettant de garantir une minimisation de l'erreur.

Le gradient est obtenu à travers le calcul des dérivées partielles de la fonction à minimiser par rapport aux variables que l'on souhaite voir évoluer, soit les poids  $W$  et le biais  $b$ ,

$$W_{i,j}^{(l)} = W_{i,j}^{(l)} - \alpha \frac{\partial E(W,b)}{\partial W_{i,j}^{(l)}}, \quad (2.5)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial E(W,b)}{\partial b_i^{(l)}}, \quad (2.6)$$

avec  $W_{i,j}^{(l)}$  le poids de connexion entre les neurones  $i$  et  $j$ , et  $b_i^{(l)}$  le biais associé au neurone  $i$  dans la couche  $l$  du réseau. Le paramètre  $\alpha$ , fixé ou fluctuant en fonction de l'avancée de l'optimisation, gère la proportionnalité du pas à effectuer dans une direction, ici opposée au gradient. Une valeur élevée du pas accélère la descente, mais au risque de rater le minimum, empêchant ainsi la descente de gradient de converger. Au contraire, une très faible valeur du pas permet de converger, mais la convergence sera plus lente. Les dérivées partielles par rapport aux poids  $W$  et biais  $b$  sont calculées à partir des erreurs propagées entre les couches. L'erreur  $\delta_i^{ol}$  de la dernière couche  $ol$  pour un neurone  $i$  se calcule comme suit,

$$\delta_i^{ol} = \frac{\partial E(W,b)}{\partial z_i^{ol}} = -(y_i - a_i^{(ol)}) f'(z_i^{(ol)}), \quad (2.7)$$

avec  $f'$  la dérivée de la fonction d'activation  $f$ . Puis les erreurs des autres couches, jusqu'à la seconde, sont calculées par rétro-propagation,

$$\delta_i^l = \left( \sum_{j=1}^{n^{(l)}} W_{j,i}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}), \quad (2.8)$$

avec  $n^{(l)}$  le nombre de neurones dans la couche  $l$ . Les dérivées partielles par rapport aux poids  $W$  et biais  $b$  pour une couche donnée  $l$  sont définies comme suit,

$$\frac{\partial E(W,b)}{\partial W_{i,j}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}, \quad (2.9)$$

$$\frac{\partial E(W,b)}{\partial b_i^{(l)}} = \delta_i^{(l+1)}. \quad (2.10)$$

L'algorithme de rétro-propagation calcule les erreurs diffusées à travers les couches pour faire converger les poids et biais afin de minimiser l'erreur entre la sortie du réseau et celle

souhaitée. Autrement dit, l'évolution des poids résulte de la minimisation de l'erreur de la fonction de coût (équation 2.4). L'inconvénient de la méthode de rétro-propagation est son temps de convergence qui dépend de la dimension des données et du pas de descente. En effet, pour obtenir le gradient exact, tous les exemples des données d'entraînement sont soumis au réseau pour apprentissage, mais cela est coûteux. Une alternative est le calcul du gradient sur moins d'exemples afin d'obtenir une approximation du gradient exact associé à l'ensemble des données. Cette alternative correspond à une descente de gradient stochastique (ou *stochastic gradient descent*, SGD) [21]. De par la nature stochastique, la recherche du gradient n'est pas directe comme la méthode initiale. Cependant, la convergence presque sûre du SGD vers le minimum global, si la fonction est convexe, est démontrée au travers d'une réduction itérative du pas de descente [22]. Par conséquent, le SGD réduit le temps nécessaire à la convergence et donc à l'apprentissage.

Les réseaux FF reposent principalement sur un apprentissage supervisé. Par exemple, le réseau convolutionnel (ou *convolutional neural network*, CNN) [92] empile des couches de propriétés différentes afin de transformer les données soumises de manière supervisée. La première transformation est une convolution de diverses formes, et la seconde correspond à des fonctions d'agrégation ou *pooling*. La succession de ces transformations extrait des caractéristiques spécifiques relatives au type de convolution (plusieurs noyaux) ou *pooling* (moyen ou maximum) souhaité. Le réseau de maximisation hiérarchique (ou *hierarchical maximisation*, HMAX) [117] [118] utilise une alternance de neurones appelés cellules simples et complexes en fonction de leurs applications. Le passage de l'information à travers le HMAX permet une extraction des caractéristiques relatives aux fonctions de ces cellules axées sur l'agrégation des données.

Il existe des réseaux FF qui extraient l'information de manière non supervisée. Par exemple l'auto-encodeur [37] cherche à reconstruire en sortie l'entrée soumise à travers l'apprentissage d'un dictionnaire de caractéristiques en couche cachée. Cependant, quelle que soit la supervision, ces réseaux sont limités quand il s'agit de comprendre la corrélation temporelle dans les données comme la compréhension du langage. Une solution consiste à utiliser des réseaux récurrents qui prennent en compte la notion de temporalité.

### 2.3.2 Récurrence et mémoire

Un réseaux récurrent (ou *recurrent neural network*, RNN) illustré par la figure 2.2b adopte l'architecture FF (figure 2.2a), mais les neurones de la couche cachée sont connectés entre eux pour associer une temporalité aux données. La récurrence recherche des corrélations entre les données apprises à des temps différents. En effet, les RNN sont principalement utilisés dans des tâches impliquant une notion temporelle comme des séries chronologiques [31] ou la reconnaissance du langage [50]. Les connexions des neurones de la couche cachée consistent en une chaîne dépendante de la récurrence souhaitée. L'apprentissage s'effectue

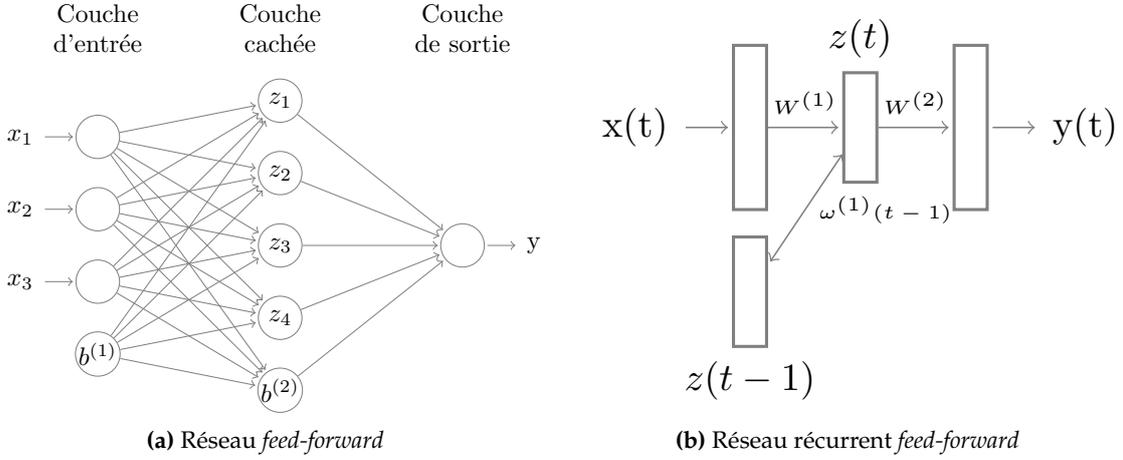


FIGURE 2.2 – Illustration de réseaux de neurones artificiels.

en déroulant les couches récurrentes comme un réseau composé de plusieurs couches et utilise une extension de la rétro-propagation à travers le temps (ou *Back Propagation Through Time*, BPTT) [137], *c.-à-d.* à travers les couches récurrentes. Cependant, le problème majeur du réseau récurrent est sa capacité à dégrader l’information apprise. En effet, l’apprentissage à l’aide de rétro-propagation consiste à corriger les déviations des poids, mais une trop forte récurrence implique un risque de voir disparaître ou exploser le gradient [13]. La disparition ou l’explosion du gradient fait référence respectivement au décroissement ou à l’accroissement dans la norme du gradient en présence de données temporairement distantes. Par conséquent, le réseau n’est pas en mesure d’apprendre les corrélations possibles.

La différence clé entre la rétro-propagation et son extension à travers le temps est que pour cette dernière, les gradients des poids  $W$  sont sommés à chaque itération de temps. En effet, les paramètres d’un réseau de neurones traditionnel ne sont pas partagés à travers les couches, ainsi la somme de l’ensemble des poids est superflue. Le BPTT correspond finalement à un BP appliqué sur un réseau récurrent déroulé comme démontré par les équations suivantes. Ainsi, dans le cas d’un réseau récurrent, l’équation (2.2) reste la même pour le calcul des activations de la couche de sortie, mais elle est modifiée pour le passage des couches cachées,

$$z_i^{(l)}(t) = \sum_{j=1}^{n^{(l)}} W_{i,j}^{(l-1)} a_j^{(l-1)}(t) + \sum_{T=1}^r \sum_{j=1}^{n^{(l)}} \omega_{i,j}^{(l-1)}(t-T) a_j^{(l-1)}(t-T) + b_i^{(l-1)}(t), \quad (2.11)$$

avec  $n^{(l)}$  le nombre de neurones dans la couche  $l$ ,  $r$  le nombre de récurrences par couche et  $\omega^{(l-1)}(t-T)$  la matrice de poids de la couche  $l-1$  à temps  $t-T$  tel que  $2 \leq l \leq ol$ .

Le calcul des gradients prend en compte la temporalité, on obtient ainsi pour les équations (2.7) et (2.8) respectivement  $\delta_i^{ol}(t)$  et  $\delta_i^l(t)$ . Désormais l’erreur de la couche de récurrence

devient

$$\delta_i^{(l)}(t-T) = \left( \sum_{j=1}^{n^{(l)}} \omega_{j,i}^{(l-1)}(t-T) \delta_j^{(l+1)}(t) \right) f'(z_i^{(l)}(t-T)). \quad (2.12)$$

Puis les matrices de poids dans la récurrence sont mises à jour suivant l'équation

$$\frac{\partial E(W,b)}{\partial \omega_{i,j}^{(l)}(t-T)} = a_j^{(l)}(t-T) \delta_i^{(l+1)}(t). \quad (2.13)$$

Les réseaux de neurones, qu'ils soient récurrents ou non, restent des compositions de fonctions. Ajouter une notion de temps aux réseaux ne fait qu'ajouter une composition supplémentaire de fonctions. Par conséquent, le BPTT devient coûteux en terme de ressources de calcul, rendant préférable la version tronquée du BPTT qui réduit les itérations nécessaires pour des données de dimension élevée. Cependant, cette troncature amène un problème d'apprentissage sur les dépendances à très long terme que ne rencontre pas un BPTT classique.

Une solution à la problématique de la diffusion du gradient est d'utiliser une mémoire pour gérer temporellement les informations, *c.-à-d.* éviter une déformation par une abstraction des données.

Un moyen de s'affranchir de la contrainte liée à la notion même de temporalité repose sur la mémorisation des séquences comme avec le réseau de mémoire à court et long terme (ou *Long Short Term Memory*, LSTM) [62]. Ce réseau de neurones évite la dispersion du gradient inhérente aux RNN classiques grâce à sa capacité d'effacer ou d'ajouter spécifiquement des données. L'altération de l'information est effectuée au travers de processus complémentaires à l'application d'une unique fonction d'activation. En effet, une régulation est appliquée sur l'information traversant une unité de calcul du LSTM appelée cellule. Pour un réseau de neurones récurrent classique, l'information transite par une unité de calcul (le neurone), à travers la combinaison linéaire de ses entrées et poids ponctuellement modifiés par une fonction d'activation (équation 2.2). En revanche, l'information dans une cellule LSTM traverse trois portes liées au passage critique de l'information, soit l'entrée (*input gate*), la gestion des données (*forget gate*) et la sortie (*output gate*). Autrement dit, la cellule prend une décision quant aux informations à garder en mémoire, et identifie quand il faut autoriser la lecture, l'écriture et l'effacement à travers l'ouverture ou la fermeture de ces portes. Pour chaque porte, la fonction d'activation sigmoïde est utilisée du fait de sa capacité à projeter les données dans un ensemble compris entre 0 et 1. Cet ensemble correspond respectivement à ne rien laisser entrer et tout laisser passer pour une cellule donnée. La fonction sigmoïde étant différentiable, elle est utilisable dans l'algorithme de rétro-propagation. Par conséquent, les portes d'une cellule agissent sur les signaux qu'elle reçoit en bloquant ou laissant passer l'information en fonction des poids associés au passage entre les portes. Ces poids évoluent durant l'apprentissage comme pour les autres réseaux récurrents via l'utilisation du BPTT. Au

final, les cellules d'un réseau LSTM apprennent quand les données sont autorisées à entrer, être modifiées ou sortir. Le réseau LSTM évite par conséquent la disparition ou explosion du gradient à l'aide de la régulation appliquée aux données dans ces unités de mémoire.

L'utilisation de cette mémoire permet au réseau LSTM d'être plus performant et d'être utilisé avec succès dans des tâches d'apprentissage telles que la détection, la reconnaissance et la compréhension du langage [38] [50] [126]. Le LSTM peut aussi étiqueter des images à partir d'une base d'apprentissage [134].

Un autre réseau utilisant une mémoire dans un contexte de classification est le *Deep Belief Network* (DBN) [61]. Le réseau repose sur une architecture de machines de Boltzmann restreintes (ou *Restricted Boltzmann Machine*, RBM) empilées. La mémoire au sommet de l'empilement des RBM est une mémoire associative bidirectionnelle associant les résultats générés à l'aide des RBM avec une classe d'objet, et ce de manière supervisée. La mémoire associative bidirectionnelle (ou *Bidirectional associative memory*, BAM) est étudiée en détail dans la seconde partie de ce chapitre (section 2.5).

Le choix de réseaux RBM permet la création d'une fonction générative. Le DBN possède une approche différente des autres réseaux de neurones de par sa fonction générative. En effet, dans l'apprentissage automatique on distingue deux types de modèle. Le premier est de type discriminatif, c'est le cas pour les réseaux de neurones classiques tels que le CNN, HMAX et LSTM. Le second type, comme les RBM et DBN est quant à lui génératif. Par analogie avec une tâche de détermination du langage d'une personne, l'approche générative consisterait à apprendre chaque type de langage afin de déterminer à quelle langue le discours appartient. L'approche discriminative déterminerait la différence linguistique sans apprendre le langage même. Cette simplicité entraîne une utilisation plus répandue des méthodes discriminatives. En d'autres termes, les modèles génératifs recherchent les distributions de probabilités sous-jacentes permettant de générer par eux-mêmes des exemples. Les modèles discriminatifs estiment directement les probabilités postérieures et sont axés sur des tâches spécifiques. La compréhension d'un modèle génératif nous conduit à l'étude des réseaux récurrents à base d'énergie.

### 2.3.3 Réseaux récurrents à base d'énergie

L'apprentissage du DBN repose sur la divergence contrastive et non sur l'algorithme de rétro-propagation. Puisqu'il recherche les distributions de probabilités et non directement les estimateurs, le DBN possède une architecture qui s'appuie sur des machines de Boltzmann restreintes.

### 2.3.3.1 Machines de Boltzmann

Le modèle général d'une machine de Boltzmann (ou *Boltzmann machine*, BM) [3], est un réseau récurrent binaire à deux couches comme une architecture *feed-forward* illustrée par la figure 2.3 (gauche). La différence avec des réseaux comme le CNN est l'état stochastique, la représentation sous forme de variables aléatoires des neurones visibles  $v \in \{0,1\}^D$  et cachés  $h \in \{0,1\}^P$  appartenant respectivement aux couches d'entrée/sortie et aux couches cachées. L'état du réseau repose sur une fonction d'énergie définie comme

$$E(v,h;\theta) = -\frac{1}{2}v^T L v - \frac{1}{2}h^T J h - v^T W h, \quad (2.14)$$

où  $\theta = \{W,L,J\}$  est l'ensemble des paramètres de poids connectant les couches, et les neurones dans chaque couche. L'usage de trois matrices de poids pour deux couches permet le stockage des poids connectant les couches et les connexions neuronales au sein d'une même couche. L'état stochastique des neurones induit une distribution de probabilité sur ces unités relative à la fonction d'énergie. La distribution est définie par

$$p(v,h;\theta) = \frac{\exp(-E(v,h;\theta))}{Z(\theta)}, \quad (2.15)$$

avec

$$Z = \sum_v \sum_h \exp(-E(v,h;\theta)) \quad (2.16)$$

la fonction de partition sur l'ensemble des configurations possibles. La distribution de probabilité d'une unité visible de la première couche correspond à la somme par rapport aux configurations possibles des neurones de la couche cachée, soit

$$p(v;\theta) = \frac{\sum_h \exp(-E(v,h;\theta))}{Z(\theta)}. \quad (2.17)$$

Les distributions conditionnelles des unités cachées et visibles sont définies par

$$P(v_i = 1 | h, v_{-i}) = f \left( \sum_{j=1}^P W_{i,j} h_j + \sum_{k=1/i}^D L_{i,k} v_k \right), \quad (2.18)$$

et

$$P(h_j = 1 | v, h_{-j}) = f \left( \sum_{i=1}^D W_{i,j} v_i + \sum_{m=1/j}^P J_{j,m} h_m \right), \quad (2.19)$$

avec  $f$  la fonction d'activation des neurones, la plus courante étant la fonction sigmoïde listée dans le tableau 2.1. La notation  $v_{-i}$  correspond à prendre toutes les valeurs de  $v$  sauf sa  $i^{\text{ème}}$  valeur.

Une machine de Boltzmann restreinte (RBM) sur laquelle repose le DBN est identique au modèle BM, à la différence près que les neurones d'une même couche ne sont pas connectés

entre eux. Par conséquent, la formulation de la fonction d’énergie associée se réécrit sous la forme

$$E(v,h;\theta) = -b^T v - c^T h - h^T W v, \quad (2.20)$$

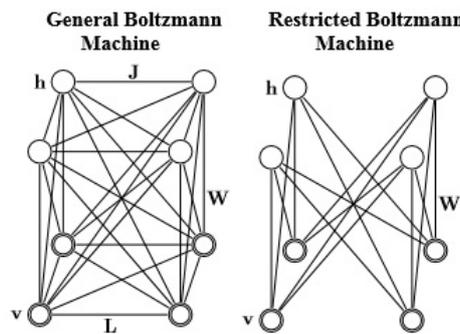
où il ne reste que les poids de connexion  $W$  entre les couches. Les paramètres  $b$  et  $c$  sont les biais respectifs des couches visibles et cachées. Les probabilités conditionnelles des neurones sont aussi modifiées, les équations (2.18) et (2.19) deviennent respectivement

$$P(v_i = 1|h) = \sigma(b_i + W_i^T h), \quad (2.21)$$

et

$$P(h_j = 1|v) = \sigma(c_j + W_j^T v). \quad (2.22)$$

La figure 2.3 illustre la différence de structure entre une machine de Boltzmann classique (à gauche) et une restreinte (à droite).



**FIGURE 2.3** – Différence de structure entre deux types de machine de Boltzmann. Figure issue de l’article Deep Boltzmann Machine [114].

La méthode d’apprentissage est différente de celle utilisant l’algorithme de rétro-propagation du fait de l’état stochastique des neurones. L’évolution des poids est effectuée en maximisant la vraisemblance logarithmique de l’état des neurones en fonction des données traitées, soit

$$\log(p(v;\theta)) = \log \left( \sum_h \exp(-E(v,h;\theta)) \right) - \log \left( \sum_{\hat{v}} p(\hat{v}) \sum_h \exp(-E(\hat{v},h;\theta)) \right). \quad (2.23)$$

L’optimisation est faite au travers d’une descente de gradient stochastique. Ainsi, l’évolution des poids s’appuie sur le calcul de la dérivée partielle de la vraisemblance logarithmique négative définie par

$$\frac{\partial \log(p(v;\theta))}{\partial \theta} = - \frac{\partial \log(\sum_h \exp(-E(v,h;\theta)))}{\partial \theta} + \sum_{\hat{v}} p(\hat{v}) \frac{\partial \log(\sum_h \exp(-E(\hat{v},h;\theta)))}{\partial \theta}. \quad (2.24)$$

Les deux termes de l’équation précédente se réfèrent respectivement à la phase positive et négative de l’apprentissage. Les noms des phases ne dépendent pas de leurs signes, mais

de leurs impacts sur la densité de probabilité définie par le modèle. Le premier terme accroît la probabilité des données de l'entraînement, en réduisant l'énergie correspondante  $-\log \sum_h \exp(-E(v,h;\theta))$ , c'est l'espérance dépendante des données. Le second terme décroît la probabilité des échantillons générés par le modèle, c'est l'espérance du modèle. Le gradient et donc l'erreur résultent de la différence des espérances relatives aux données et au modèle.

Bien que le premier terme soit facilement calculable, le second nécessite une approche spécifique du fait du calcul de probabilité  $p(\hat{v})$ . L'échantillonnage de  $p(x)$  est fait par la méthode d'échantillonnage de Gibbs pour obtenir un estimateur du gradient de la vraisemblance logarithmique sous plusieurs itérations, le tout permettant de mettre à jour les poids du réseau. Ces étapes sont utilisées pour avoir une idée de ce que le modèle peut capturer de la distribution des données. La distribution des neurones de la couche cachée  $h$  est obtenue par échantillonnage en connaissant les données  $x$ , puis un nouveau  $x$  conditionnellement à  $h$  est calculé. L'échantillonnage de Gibbs pour  $k$  itérations à partir d'un exemple d'apprentissage  $\hat{P}$  est le suivant,

$$\begin{aligned}
 x_1 &\sim \hat{P}(x) \\
 h_1 &\sim P(h|x_1) \\
 x_2 &\sim P(x|h_1) \\
 h_2 &\sim P(h|x_2) \\
 &\vdots \\
 x_{k+1} &\sim P(x|h_k).
 \end{aligned} \tag{2.25}$$

Cependant, la convergence d'un tel échantillonnage est coûteuse. Par conséquent, les modèles plus récents s'appuient sur une alternative à la méthode de Gibbs, avec un procédé de Monte Carlo combiné à l'utilisation de chaînes de Markov (MCMC). Cette méthode MCMC calcule une approximation du gradient pour chaque sous itération associée au couple  $(v,h)$  à chaque itération d'une chaîne de Markov. Une dernière méthode basée sur le MCMC est la divergence contrastive [60].

En théorie, nous devrions utiliser une solution numérique pour évaluer la fonction d'énergie du RBM (équation 2.20). La solution serait obtenue par utilisation de la différence finie pour calculer le gradient à un point donné de l'espace des paramètres, puis de la descente de gradient afin de trouver un minimum local. Cependant, pour des données à dimension élevée, l'intégration de la notion de temps est handicapante, ce qui est le cas avec les RBM. En d'autres termes, il s'agit d'une tentative de minimiser une fonction d'énergie non évaluable. Bien que la fonction d'énergie ne soit pas évaluable, la méthode de divergence contrastive (CD) estime le gradient de cette fonction. La CD implique l'utilisation combinée d'une méthode de Monte Carlo et de chaînes de Markov (MCMC). Il est montré empiriquement qu'une seule itération est suffisante pour approcher le gradient [60].

### 2.3.3.2 Divergence Contrastive

Les échantillons de distribution ne peuvent pas être directement tirés de  $p(v; \theta)$  puisque la valeur de la fonction de partition est inconnue. Cependant, un échantillonnage via la méthode MCMC transforme les données d’apprentissage (tirées de la distribution souhaitée) en données de la distribution proposée. En effet, la transformation implique seulement le calcul du ratio de deux probabilités,  $p(v'; \theta) / p(v; \theta)$ , annulant la fonction de partition.

Par exemple si le CD est utilisé par le RBM, la distribution de probabilité

$$p(v; W) = \frac{\exp(-E(v; W))}{Z(W)}, \quad (2.26)$$

où

$$Z = \sum_v \exp(-E(v; W)), \quad (2.27)$$

est une constante de normalisation et  $E(v; W)$  la fonction d’énergie, la vraisemblance logarithmique de l’équation (2.24) se réécrit

$$\begin{aligned} L(W; \chi) &= \frac{1}{N} \sum_{n=1}^N \log p(x_n; W) \\ &= \langle \log p(x; W) \rangle_0 \\ &= - \langle E(x; W) \rangle_0 - \log(Z(W)), \end{aligned} \quad (2.28)$$

où  $\langle \cdot \rangle_0$  désigne l’espérance avec respect de la distribution des données d’apprentissage. Le gradient se calcul alors

$$\frac{\partial L(W; \chi)}{\partial W} = - \left\langle \frac{\partial E(x; W)}{\partial W} \right\rangle_0 + \left\langle \frac{\partial E(x; W)}{\partial W} \right\rangle_\infty \quad (2.29)$$

où  $\langle \cdot \rangle_\infty$  désigne l’espérance avec respect de la distribution générée par le modèle

$$p_\infty(x; W) = p(x; W). \quad (2.30)$$

L’espérance  $\langle \cdot \rangle_0$  est aisément calculée à partir des données échantillonnées  $\chi$ . Cependant, l’espérance  $\langle \cdot \rangle_\infty$  implique le calcul de la constante de normalisation  $Z(W)$  qui n’est généralement pas calculée efficacement compte tenu de la somme d’un nombre exponentiel de termes.

L’approche usuelle consiste à employer une chaîne de Markov pour approcher la moyenne d’une distribution avec une moyenne des échantillons de  $p_\infty(x; W)$  afin de converger vers  $p(x; W)$  au travers des chaînes jusqu’à un équilibre. L’avantage de cette méthode MCMC est d’être aisément applicable à plusieurs classes de distribution de  $p(x; W)$ . Cependant, un désavantage est le temps nécessaire pour atteindre cet équilibre compte tenu du grand nombre d’itérations nécessaires et de la difficulté à déterminer si l’équilibre a été atteint. Par ailleurs, un autre défaut est la grande variance du gradient.

Afin d'éviter la complexité du calcul du gradient de la vraisemblance logarithmique, la divergence contrastive suit approximativement le gradient d'une fonction différentiable. L'apprentissage du maximum de vraisemblance minimise la divergence de Kullback Leibler des distributions liées aux données et au modèle,

$$KL(p_0||p_\infty) = \sum_x p_0(x) \log \frac{p_0(x)}{p(x; W)}. \quad (2.31)$$

Puis l'apprentissage par divergence contrastive suit approximativement le gradient de la différence de deux divergences [60],

$$CD_n = KL(p_0||p_\infty) - KL(p_n||p_\infty). \quad (2.32)$$

L'apprentissage par CD consiste à lancer une chaîne de Markov à la distribution des données  $p_0$  et à parcourir la chaîne pour un faible nombre d'itérations. La divergence contrastive se révèle efficace dans divers problèmes exigeant un échantillonnage de Gibbs ou une méthode hybride de Monte Carlo en tant qu'opérateur de transition pour une chaîne de Markov.

Il existe une version améliorée, la divergence contrastive persistante (ou *Persistent Contrastive Divergence*, PCD). L'approximation via PCD est obtenue par CD en remplaçant l'échantillon  $v^{(k)}$  par un échantillon produit par une chaîne de Gibbs indépendante de l'échantillon  $v^{(0)}$  de la distribution d'entraînement [42]. L'idée fondamentale sous-jacente du PCD s'appuie sur la proximité d'une chaîne de Gibbs à une distribution stationnaire si le taux d'apprentissage est suffisamment petit. Par conséquent, la variation du modèle est minimale entre les mises à jour des paramètres.

L'algorithme 1 démontre que l'apprentissage d'un réseau de neurones repose sur le calcul du gradient des poids et biais afin de satisfaire la minimisation de la fonction de coût. Cette dernière définit le type de modèle du réseau (discriminatif ou génératif) impliquant respectivement l'usage de l'algorithme de rétro-propagation ou de divergence contrastive. Cependant, quel que soit le type de modèle, l'apprentissage de ces réseaux de neurones reste coûteux, et ce d'autant plus que le nombre de neurones par réseau et la quantité de données à traiter s'accroissent. Il est indispensable de simplifier les calculs afin de répondre à l'usage des réseaux de neurones dans des applications toujours plus variées, et disponibles sur un plus grand panel de supports de calcul.

## 2.4 Vers une simplification des calculs

Désormais les réseaux *feed-forward* sont utilisés en apprentissage profond (DNN) [90] [93] consistant à empiler successivement des couches de neurones. La méthode DNN repose principalement sur les réseaux convolutionnels (CNN) dont l'utilisation est prédominante

**Algorithme 1** Phase d’apprentissage en fonction du type de réseau

<b>Discriminatif</b>	<b>Génératif</b>
Fonction à minimiser :	Énergie du réseau :
1: $\operatorname{argmin}_{W,b} E(W,b) = \frac{1}{2} \ y - y_{W,b}(x)\ ^2$ .	1: $E(v,h;\theta) = -\frac{1}{2}v^T Lv - \frac{1}{2}h^T Jh - v^T Wh$ ,
2: $\left[ \frac{\partial E}{\partial W^l}, \frac{\partial E}{\partial b^l} \mid \frac{\partial E}{\partial \omega^l(t-T)} \right] =$ Rétro-propagation à travers le temps.	Fonction à minimiser :
3: $W^{(l)} = W^{(l)} - \alpha \frac{\partial E}{\partial W^l}$	2:
4: $b^{(l)} = b^{(l)} - \alpha \frac{\partial E}{\partial b^l}$	$\operatorname{argmax}_{\theta} \log(p(v,\theta))$
5: $\omega^{(l)}(t-T) = \omega^{(l)}(t-T) - \alpha \frac{\partial E}{\partial \omega^{(l)}(t-T)}$ (BPTT)	$= \log \left( \sum_h \exp(-E(v,h;\theta)) \right)$
	$- \log \left( \sum_{\hat{v}} p(\hat{v}) \sum_h \exp(-E(\hat{v},h;\theta)) \right)$
	(2.33)
	3: $\frac{\partial \log(p(v;\theta))}{\partial \theta} =$ Divergence contrastive
	4: $\theta = \theta - \alpha \frac{\partial L(\theta)}{\partial \theta}$

dans les logiciels mis à disposition gratuitement tels que TensorFlow [1], Theano [12] [14], Torch [30] ou Caffe [77] pour ne citer que les principaux.

L’usage de ces réseaux s’inscrit dans un contexte de classification, de détection [10] [116] [125] et reconnaissance d’objets [129] [143] ou de régions dans les images. Ces tâches s’effectuent de manière supervisée et lorsque les données d’apprentissage sont abondantes, comme pour la reconnaissance de formes dans le jeu de Go [120], la détection et reconnaissance de visages [47] [127] ou de textes dans des images naturelles [36]. Puisque les images peuvent être étiquetées directement à partir des pixels, ce type de reconnaissance peut être utilisé par des véhicules dans une optique de conduite autonome [54].

Le succès de ces architectures s’explique notamment par l’utilisation des régularisations qui rendent les réseaux plus efficaces en généralisant au-delà des données d’apprentissage. Outre la méthode d’abandon (ou *dropout*) [123], permettant au réseau d’abandonner aléatoirement des connexions afin d’empêcher un sur-apprentissage des données, l’utilisation de la fonction d’activation d’unités linéaires rectifiées (ou *Rectified linear units*, ReLU) est devenue primordiale [34][102]. Cependant, les performances obtenues doivent beaucoup à l’efficacité des cartes graphiques (GPU) et à la possibilité de créer des étiquettes supplémentaires en augmentant la base d’apprentissage.

Une contrainte des réseaux de neurones optimisés réside dans leur besoin élevé de ressources en terme de puissance, d’énergie et de temps lors de la phase d’apprentissage [66]. De plus, avec l’avènement des réseaux profonds (DNN), la demande en ressources augmente avec le nombre de couches du réseau, et donc du nombre de neurones disponibles

dans l'ensemble du réseau. Désormais, les réseaux profonds sont principalement entraînés sur des cartes graphiques devenues depuis peu une ressource de calcul moins onéreuse [29].

L'usage de ces cartes graphiques limite l'implantation des DNN sur des environnements embarqués tels que des puces à faible consommation ou des *smartphones*. Le défi est désormais de faire évoluer les réseaux afin de les porter sur des supports nécessitant moins de ressources. Une évolution récente est l'émergence des réseaux de neurones composés de poids binaires afin de simplifier les calculs durant l'apprentissage [33]. Une solution alternative que nous explorons dans cette thèse, est l'emploi de réseaux de neurones conceptualisés sous forme de mémoires. En effet, dans un contexte de classification, un réseau de neurones optimisé (à travers ses couches cachées) crée un nouvel espace afin de classer linéairement des variables non séparables. Ce nouvel espace est défini par la transformation de la dimension des données en caractéristiques spécifiques afin de maximiser leurs classifications.

L'intégration d'une mémoire dans ce type de réseau permet d'utiliser la temporalité en tant que notion discriminante supplémentaire dans les données. En revanche, dans les mémoires associatives, *c.-à-d.* des réseaux devenant eux-mêmes une mémoire, le changement de représentation des données correspond à un plan où toutes les informations stockées forment des bassins d'attraction. Le paradigme de la classification se trouve modifié par ce changement d'architecture. La classification par les mémoires associatives, de par leurs bassins d'attraction, se rapproche de la méthode des plus proches voisins. Les données sont codées sans altérations dans le réseau. Par conséquent, les mémoires associatives sont principalement basées sur des architectures binaires réduisant l'impact de la mémoire nécessaire pour la phase d'apprentissage. La mémoire est notamment sollicitée durant la phase de récupération afin de retrouver un objet, mais son architecture binaire réduit le coût des calculs. La récupération est accomplie de manière itérative à l'instar de la phase d'apprentissage d'un réseau optimisé.

Afin de bien illustrer la distinction entre un réseau de neurones optimisé et une mémoire associative, nous présentons ici un exemple. Nous définissons l'espace des données initial comme une chambre d'adolescent lambda que nous considérons comme chaotique. Les données sont des objets qui y sont présents tels qu'un lit, des meubles, vêtements, livres, déchets, etc. L'usage des réseaux optimisés consiste en premier lieu à ranger la chambre afin que chaque objet soit discernable. Au travers du rangement, nous créons un espace linéairement séparable. Ce rangement nécessite du temps et de l'énergie, mais une fois effectué, le futur rangement n'en est que plus simplifié. Au contraire, l'usage d'une mémoire associative consiste à laisser telle quelle la chambre et mémoriser l'emplacement des objets créant leurs bassins d'attraction associés, *c.-à-d.* qu'un objet pourra attirer l'œil plus qu'un autre. Aucun rangement n'est nécessaire, tout est là, et c'est la recherche d'un objet qui requiert du temps et de l'énergie. La question à se poser est la suivante : à l'ouverture de la chambre dans son état initial, si on souhaite rechercher un objet, quelle est la méthode qui nécessite le moins

de ressources pour un résultat similaire ?

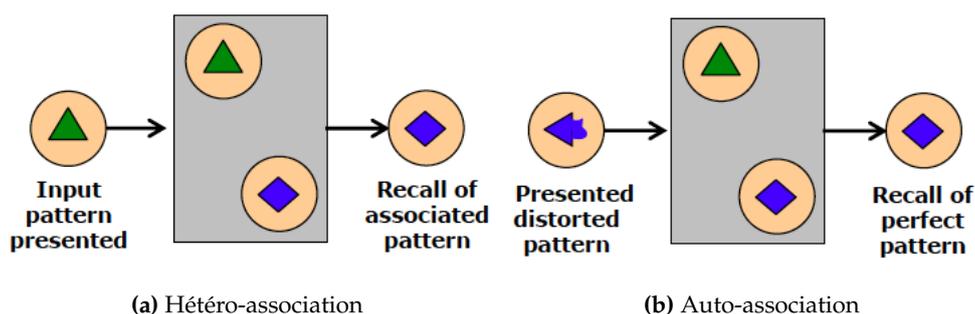
Nous allons maintenant entrer dans les détails de ces mémoires associatives afin de justifier notre utilisation des réseaux à cliques.

## 2.5 Mémoires associatives

### 2.5.1 Définition

La mémoire est un processus permettant le codage de l'information afin de la stocker et de pouvoir la retrouver [11]. Le codage transforme l'information reçue sous forme interprétable par la mémoire. Le stockage permet d'enregistrer au sein de la mémoire l'information. La récupération permet de rappeler une information apprise suite à une stimulation. L'idée d'association dans la mémoire est d'abord approchée d'un point de vue philosophique (Aristote) et psychologique (David Hume) avant que la science ne cherche à identifier le fonctionnement d'une telle mémoire associative au sein du cerveau humain [81].

Un exemple d'une association faite pas le cerveau est celle d'un nom avec le visage (ou inversement) d'une personne [136]. C'est l'hétéro-association (figure 2.4a) puisque deux entités différentes sont associées. Un autre exemple est la complétion, qu'il s'agisse de mots dans un texte à trous ou une image. Si le texte ou l'image sont mémorisés au préalable, la complétion récupère l'information initialement apprise, c'est l'auto-association (figure 2.4b). La complétion d'une information non connue à l'aide de ce qui a été mémorisé correspond à un cas de généralisation.



**FIGURE 2.4** – Types de mémoires associatives. Figure issue du manuscrit Artificial neural networks [55].

Une mémoire associative neuronale (ou *Neural Associative Memory*, NAM) est définie par le fait que le réseau de neurones apprend une association à l'aide de la matrice de poids  $W$  [109]. Cette matrice  $W$  stocke les connexions entre deux couches de neurones, *c.-à-d.* une

entrée et une sortie pour une mémoire de type hétéro-associative. Dans le cas d'une mémoire auto-associative, les connexions sont récurrentes entre les neurones d'une seule et même couche.

### 2.5.2 Mémoires associatives neuronales (NAM)

Une définition du formalisme classique de ces NAM est donnée dans les mémoires de Steinbuch [124], *lernmatrix*, qui peuvent être vues comme un cluster d'unités modélisées pour ressembler aux caractéristiques d'un neurone biologique. Steinbuch a mis au point une architecture permettant l'apprentissage de mots de code au travers d'une représentation matricielle des connexions entre neurones. Les mémoires holographiques [45] [95], qui conduiront au développement des premiers modèles de réseaux de neurones à mémoires associatives [5] [8] [9], s'appuient sur le concept introduit par Steinbuch.

L'apprentissage pour les NAM s'effectue avec le stockage dans une matrice de poids  $W$  de l'encodage des  $p$  associations des paires entrées-sorties  $(x^k, y^k)$  pour  $k = 1, \dots, p$ . La matrice  $W$  est définie selon les équations suivantes correspondant à une règle d'apprentissage de type additive

$$W_{i,j}^k = x_i^k y_j^k, \quad (2.34)$$

$$W = \alpha \sum_{k=1}^p W^k, \quad (2.35)$$

avec  $\alpha$  un paramètre de proportion ou de normalisation. On note que pour les mémoires auto-associatives, les paires représentent le même objet, soit  $y^k = x^k$ . Par conséquent, l'auto-association induit une matrice symétrique des connexions appelée matrice d'adjacence.

L'intérêt des mémoires associatives réside en leur phase de décodage qui permet de retrouver, à partir d'un morceau du signal d'entrée  $x$  et à l'aide d'une fonction de seuil, un message préalablement appris

$$y_j = \begin{cases} 1, & \text{si } u_j \geq \theta_j \text{ avec } u_j = \sum_i x_i W_{i,j}, \\ 0, & \text{sinon.} \end{cases} \quad (2.36)$$

S'appuyant sur ce principe d'encodage et de décodage, un des premiers et plus simples modèles de mémoire associative est l'associateur linéaire (AL) [8] [103] fortement inspiré du modèle du Perceptron. Le réseau, illustré par la figure 2.5a, est aussi connu sous le nom de mémoire par matrice de corrélation [85]. Le AL est un réseau de type *feed-forward* pour achever la phase de décodage en une seule itération par application d'une règle de seuil (équation 2.36). Contrairement à la première architecture de mémoire associative *lernmatrix*, les espaces d'entrée et de sortie de l'associateur linéaire ne sont pas limités à des vecteurs

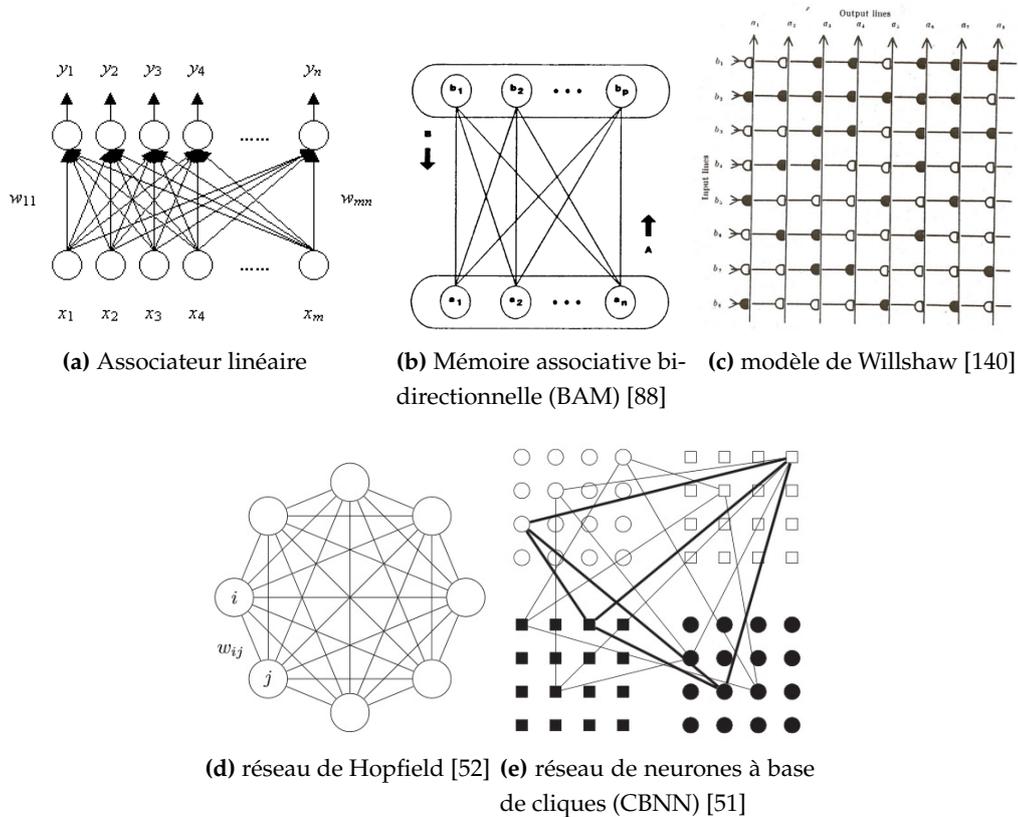


FIGURE 2.5 – Architectures de mémoires associatives.

binaires de la forme  $x \in \{0;1\}^n$  ou  $y \in \{0;1\}^m$ , mais peuvent être composés de valeurs réelles dans  $\mathbb{R}^n$  ou  $\mathbb{R}^m$  [104].

La règle d'apprentissage de Hebb avec des vecteurs binaires  $\{0;1\}$  est étudiée avec une première analyse asymptotique des poids via le réseau de Willshaw [138]. Le modèle initial de Willshaw est de type hétéro-associatif et, à l'instar de l'associateur linéaire, il s'appuie sur une matrice de corrélation. Cependant, la règle d'apprentissage pour le réseau de Willshaw contraint la matrice de poids  $W$  à rester binaire à l'aide d'un seuillage ou *clipping*,

$$W_{i,j} = \begin{cases} 1, & \text{si } W_{i,j} > 0 \\ 0 & \text{sinon.} \end{cases} \quad (2.37)$$

La règle du seuillage est aussi utilisée par les autres réseaux tels que celui de Hopfield, et elle peut être vue comme l'utilisation de la fonction signe sur la matrice de poids. L'équation (2.37) correspond au cas binaire, mais la règle s'applique aussi au cas bipolaire  $\{-1;1\}$ .

L'utilisation alternative des espaces d'entrée et de sortie sous la forme bipolaire est introduite par le réseau de Hopfield [63]. La mémoire associative de Hopfield est de type auto-

associative, avec l'utilisation de récurrence au sein des neurones composant le réseau. Son architecture est illustrée par la figure 2.5d. Sa phase de décodage itérative repose sur une fonction d'énergie. Elle représente l'état du réseau, et son optimisation permet la convergence du réseau vers un état d'équilibre par changement local et asynchrone des neurones. Puisque le modèle est auto-associatif, l'entrée et la sortie sont identiques ( $x = y$ ) résultant en une matrice de poids  $W$  symétrique telle que  $W_{i,j} = W_{j,i}$ . De plus, puisque les neurones ne se connectent pas à eux-mêmes, l'équation (2.34) pour ce modèle devient

$$W_{i,j}^k = \begin{cases} x_i^k y_j^k, & \text{si } i \neq j \\ 0, & \text{sinon.} \end{cases} \quad (2.38)$$

En ce qui concerne la phase de décodage, c'est le changement d'état des neurones de manière asynchrone (ou synchrone au risque de réduire les performances) reposant sur un ordre aléatoire ou défini au préalable qui assure une convergence du système vers un minimum local [6] [63]. Par conséquent, l'équation (2.36) devient pour le cas discret

$$y_i = \begin{cases} 1, & \text{si } \sum_{j=1}^n W_{i,j} x_j > \theta_i \\ -1, & \text{si } \sum_{j=1}^n W_{i,j} x_j < \theta_i \\ x_i & \text{sinon.} \end{cases} \quad (2.39)$$

Une version continue utilise des fonctions bornées et continues telles qu'une sigmoïde ou tangente hyperbolique [64]. Le terme d'énergie représente l'état du réseau à un instant  $t$ , et il est défini par une fonction de Lyapunov, dont l'équation est la suivante :

$$E_{hop} = \sum_{i=1}^n \sum_{j=1}^n W_{i,j} x_i x_j + 2 \sum_{i=1}^n \theta_i x_i. \quad (2.40)$$

Une analyse de la borne inférieure de la fonction d'énergie montre une décroissance monotone garantissant la convergence du réseau vers un minimum local, et donc un état stable dès lors que l'entrée  $x_i$  change, c'est la phase de décodage [6] [63]. L'utilisation d'une fonction d'énergie offre l'avantage de permettre l'interprétation des minima comme représentants des bassins d'attraction.

L'utilisation de la règle additive (équation 2.38) pour le modèle de Hopfield fait de ce dernier une mémoire inefficace, à cause des valeurs bipolaires des neurones. En effet, la mémoire change à chaque message appris. Par conséquent, la récurrence peut annuler la connexion entre deux neurones.

Une extension pour relâcher la contrainte auto-associative du modèle de Hopfield et obtenir une mémoire hétéro-associative est proposée sous le nom de mémoires associatives

bidirectionnelles (BAM)[87] [88]. Tout comme celle d’Hopfield, l’architecture est dotée d’unités binaires ou bipolaires pour la version discrète, mais peut utiliser des valeurs réelles dans le cas continu.

L’architecture hétéro-associative du réseau induit une structure composée de deux couches de neurones comme l’associateur linéaire. L’architecture du BAM est illustrée par la figure 2.5b. Sa phase d’encodage consiste à sommer les matrices de corrélation *c.-à-d.* les équations (2.34) et (2.35) entre les paires d’associations à apprendre. A l’instar de l’associateur linéaire, l’information passe d’une couche à l’autre à l’aide de la matrice de poids  $W$ . La particularité du modèle BAM réside dans la transition bidirectionnelle de l’information à l’aide de sa matrice de passage associé  $W$  et sa transposée.

La phase de récupération est effectuée suivant un décodage identique au réseau de Hopfield selon l’équation (2.39) sur chacune des deux couches par passage de l’une à l’autre jusqu’à convergence. L’équation (2.39), dans l’espace bipolaire, adaptée au modèle BAM devient

$$x_i^{new} = \begin{cases} 1, & \text{si } \sum_{j=1}^m W_{i,j} y_j^{old} > \theta_i \\ -1, & \text{si } \sum_{j=1}^m W_{i,j} y_j^{old} < \theta_i \\ x_i^{old} & \text{sinon,} \end{cases} \quad (2.41)$$

et

$$y_j^{new} = \begin{cases} 1, & \text{si } \sum_{i=1}^n W_{i,j} x_i^{old} > \theta_j \\ -1, & \text{si } \sum_{i=1}^n W_{i,j} x_i^{old} < \theta_j \\ y_j^{old} & \text{sinon.} \end{cases} \quad (2.42)$$

Tout comme le modèle de Hopfield, l’état d’un réseau BAM est représenté par une fonction d’énergie définie par

$$E_{bam}(A,B) = - \sum_{i=1}^n \sum_{j=1}^m x_i W_{i,j} y_j. \quad (2.43)$$

Le réseau de neurones de Gripon Berrou (ou Gripon Berrou Neural Network, GBNN rebaptisé CbNN) est un réseau à cliques possédant une plus large diversité de stockage que le réseau de Hopfield. Il s’appuie sur la théorie des codes correcteurs [52]. De même que son prédécesseur, le CbNN est auto-associatif, et implique une matrice d’adjacence  $W$ . Ce réseau a la particularité d’accentuer la parcimonie d’un message appris sous forme de clique. C’est un graphe complet qui connecte des groupes ou clusters de neurones en activant uniquement une seule unité par groupe [52]. Son architecture est illustrée par la figure 2.5e. L’encodage est réalisé, à l’instar des autres mémoires associatives, au travers de la somme

des matrices de corrélations (équations 2.34 et 2.35). L'apprentissage d'un CbNN repose sur la même contrainte que celle du modèle de Willshaw, avec le seuillage binaire (équation 2.37).

L'architecture auto-associative du réseau amène une phase de décodage itérative alternant deux étapes. En effet, l'activation des neurones s'appuie sur un score correspondant à la somme de leurs connexions selon l'entrée soumise. La première étape calcule les sommes, puis la seconde sélectionne les neurones à activer. L'architecture du CbNN contraint le réseau à maintenir au maximum un neurone actif par cluster. La contrainte implique que le paramètre de seuil  $\theta$  durant la phase de décodage (équation 2.39), appliqué aux scores issus du produit matriciel entre les données et la matrice d'adjacence (étape 1) doit varier. En effet, le paramètre  $\theta$  varie par l'utilisation de la fonction max au sein de chaque cluster afin de sélectionner le neurone devant être actif (étape 2). C'est la règle du gagnant prend tout (ou *Winner-take-all*, WTA). Le réseau est étudié plus en détails dans le chapitre suivant.

Nous justifions l'utilisation du réseau CbNN dans la section suivante en définissant au préalable les critères de performance des mémoires associatives présentées ici.

### 2.5.3 Critères de performance pour un NAM

Une mesure de performance populaire d'une mémoire associative est sa capacité de stockage des informations, qui est définie par la quantité d'informations contenues dans la mémoire rapportée au nombre de neurones. Cependant cette mesure ignore une propriété importante d'un modèle, le nombre de bits requis pour représenter le poids d'un neurone. Par conséquent, une autre proposition consiste à mesurer l'efficacité du réseau en divisant sa capacité par le nombre minimal de bits requis par synapse [107] [122]. Autrement dit, puisque la mémoire est représentée par une matrice d'adjacence, un premier critère de performance est calculé à partir du nombre de bits d'information contenus dans une entrée de cette matrice.

#### 2.5.3.1 Efficacité du stockage de l'information

L'efficacité s'appuie sur où et comment stocker l'information au sein du réseau. En effet, en fonction de la règle d'apprentissage utilisée, avec seuillage (équation 2.37) ou additive (équation 2.35), l'information contenue dans chaque synapse sera différente. Déterminée par Willshaw sur son réseau seuillé, la capacité optimale pour chaque neurone est asymptotique en augmentant la taille de la matrice de poids  $W$  [138]. A travers un encodage binaire des messages parcimonieux, la synapse contiendra jusqu'à  $C = \log(2) \approx 0,69$  bit par synapse active (bps) [110]. En revanche, en utilisant la règle additive (équation 2.35), la capacité monte à  $C = 1/(2\log(2)) \approx 0,72$  bps [108]. Ces capacités sont calculées pour des mémoires de type hétéro-associatives, et sont aussi valables pour des réseaux bidirectionnels [121]. Quant aux

mémoires de type auto-associative, la capacité d’information est divisée par deux du fait de la symétrie de la matrice de poids associée à ces réseaux.

La valeur d’information pour une synapse active est relative à celles des autres, et par conséquent est inhérente aux types de données à apprendre. En effet, l’apprentissage de motifs non parcimonieux décroît la quantité d’information synaptique à  $C = 0,14$  bps ou  $C = 0,33$  bps si le réseau est de type hétéro-associatif [84]. Au-delà du souhait qu’une mémoire puisse stocker un maximum de bits d’information par synapse, l’intérêt reste de pouvoir récupérer un message de la mémoire idéalement sans dégradation. Le critère primordial d’une mémoire associative est donc sa capacité critique.

### 2.5.3.2 Capacité critique

Un problème des mémoires associatives est lié à la limite d’information que le réseau peut apprendre. En effet, si la mémoire est saturée, elle peut conduire à la création de fausses informations (ou *spurious memories*). On parle ici d’ambiguïté, puisque le réseau n’est pas en mesure de sélectionner les neurones afin de récupérer avec succès un message. Une explication plus détaillée de la problématique d’ambiguïté est présentée dans la chapitre suivant.

On parle de capacité critique quand un modèle n’est plus en mesure d’apprendre des messages supplémentaires sans ajouter des erreurs durant la phase de décodage. Ces erreurs ont été largement étudiées dans le cas du modèle de Hopfield. Bien que le modèle de Hopfield converge sûrement vers un état stable, cet extremum n’est pas nécessairement global. La stabilisation de l’état du réseau vers un minimum local peut alors être considéré comme un piège [83] et ce d’autant plus que le seuil critique du réseau est atteint. La capacité critique du réseau de Hopfield a été posée à  $0,138n$  [7] avec  $n$  le nombre de neurones. Elle est listée dans le tableau 2.2 et comparée à celle des autres mémoires associatives étudiées. Les valeurs des capacités critiques représentent les bornes maximales, puisque l’étude rigoureuse de ces dernières d’un point de vue mathématique reste très rare. Une approche rigoureuse de la capacité critique du réseau de Hopfield réduit sa capacité à  $0,113n$  [41]. Autrement dit, si pour le modèle de Hopfield, qui reste le réseau de mémoire associative le plus étudié, déterminer sa capacité critique optimale est difficile, alors celles des autres réseaux que nous étudions ne peuvent être posées que comme bornes supérieures théoriques.

L’associateur linéaire stocke un nombre maximal de messages égal à la plus petite des dimensions de sa matrice de poids. Son nombre de bits d’information stockable est d’ordre  $n \log_{10}(n) / (2 \log_{10}(2))$  [101], avec  $\log_{10}$  le logarithme de la base décimale.

En termes de capacité et d’efficacité à récupérer une information dans la mémoire, le CbNN possède de meilleures performances que le modèle de Hopfield [52] ou celui de Willshaw [53]. En effet, la capacité critique de stockage du modèle de Hopfield est d’ordre  $n / (2 \log_{10}(n))$  [100] tandis que celui de Willshaw d’ordre supérieur est équivalent à celui

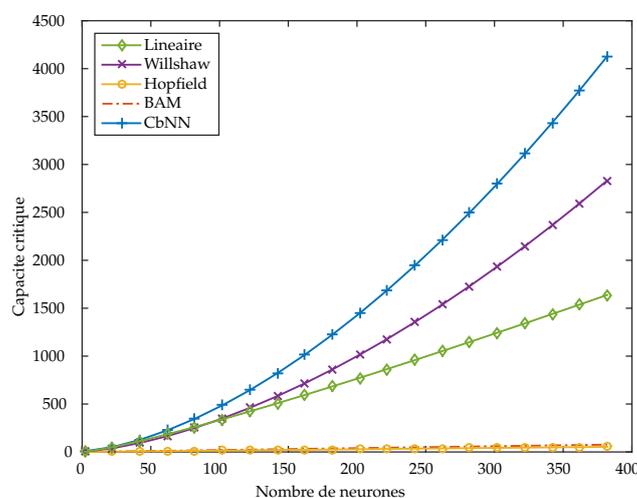


FIGURE 2.6 – Capacité critique des NAM en fonction du nombre de neurones dans le réseau.

du CbNN soit  $n^2 / (\log n)^2$ , avec  $\log$  la fonction du logarithme naturel. Bien que l'ordre des capacités critiques soit le même pour le modèle de Willshaw et le CbNN, ce dernier possède un avantage puisqu'il peut apprendre des messages de manière parcimonieuse avec l'activation d'un neurone par cluster. La figure 2.6 illustre les bornes supérieure de la capacité critique des mémoires associatives étudiées en fonction de leur nombre de neurones.

Type	Linéaire	Willshaw	Hopfield	BAM	CbNN
$M_{max}$	$\frac{n \log_{10}(n)}{2 \log_{10}(2)}$	$\frac{-n^2}{\log(n)^2} \log\left(1 - \left(\frac{\log(n)}{n}\right)^{1/\log(n)}\right)$	$0,14n < \frac{n}{2 \log_{10}(n)}$	$0,1998n$	$\frac{n^2}{\log(n)^2}$

TABLEAU 2.2 – Capacité critique des mémoires associatives

### 2.5.3.3 Avantage de la parcimonie

Afin d'augmenter la capacité critique, il est possible de réduire la densité des messages à apprendre à travers un codage parcimonieux comme identifié dans le cortex cérébral [44] [106]. Le codage parcimonieux est une minimisation du nombre de neurones mobilisés pour représenter une information. Par conséquent, le réseau opère une meilleure distinction entre deux éléments au sein de sa mémoire, tolérant ainsi l'apprentissage supplémentaire de messages. Autrement dit, la parcimonie encourage la diversité.

En traitement du signal, la notion de parcimonie correspond à maximiser le signal tout en minimisant le nombre de variables nécessaires, regroupées dans un dictionnaire, pour représenter le signal sous forme de combinaison linéaire [25] [32] [39] [40] [89]. Certains réseaux de neurones comme l'auto-encodeur parcimonieux [105] créent ces dictionnaires en regrou-

pant les caractéristiques nécessaires au codage parcimonieux de l'information. Cependant, la réduction de dimension est effectuée par minimisation, contraignant alors l'apprentissage à passer par un pré-traitement.

En revanche, avec le CbNN, la parcimonie provient de la connectivité au sein du réseau contraignant uniquement un sous-ensemble de neurones parmi ceux disponibles à être actifs (et ainsi à représenter l'information). De plus, comme avec le réseau de Willshaw [139], le CbNN est plus à même de produire un message sans erreur en se basant sur des entrées encodées de manière parcimonieuse. Ce codage réduit la complexité et le temps d'apprentissage en plus d'être biologiquement vraisemblable. En outre, le contrôle de la parcimonie [67] [68] [70] au sein du réseau gère le taux de redondance dans celui-ci et par conséquent amène une plus haute tolérance vis-à-vis des erreurs survenant durant le décodage. Autrement dit, la défaillance d'un sous-ensemble de neurones ne rend pas la représentation indécodable.

La résultante d'un tel type de codage est une forme d'orthogonalisation permettant une distinction entre diverses informations au sein du réseau. Cet effet peut être renforcé dans le cas des CbNN par l'utilisation d'un codage de Huffman [19]. D'autres réseaux se basent uniquement sur le codage parcimonieux afin de pallier les faiblesses des mémoires associatives natives [82] [98].

L'utilisation d'un codage parcimonieux avec le modèle de Hopfield repousse d'un facteur  $-\left(\frac{k}{n}\right) \log\left(\frac{k}{n}\right)^{-1}$  le seuil de sa capacité critique d'information [48]. En contraignant l'activation de  $k$  neurones parmi  $n$ , son nombre de messages maximal devient  $-n^2 / (k \log(k/n))$ . Le ratio  $k/n$  devient optimal à l'ordre  $10^{-7}$  [23].

#### 2.5.4 Application des NAM

L'emploi des mémoires associatives reste majoritairement contraint à des développements théoriques avec une validation sur des données générées aléatoirement, suivant des distributions différentes. En effet, cela permet de démontrer la faisabilité d'une tâche de reconstruction étant donnée une architecture spécifique.

Le modèle de Hopfield voit ses applications se diriger vers des problèmes d'optimisation à l'aide de sa fonction d'énergie à minimiser. En effet, le réseau sous forme de mémoire exclusive est inefficace. L'exemple le plus connu est le problème du voyageur de commerce consistant à amener le réseau à converger vers le motif le plus approprié pour le voyageur afin de visiter l'ensemble des villes [65] [141].

Dans le contexte du traitement du signal, la faible performance de stockage et le peu d'applications pratiques du modèle de Hopfield sont des freins à son usage, sauf dans le cas d'une structure impliquant un apprentissage sur des données raffinées au préalable, tel qu'un encodage parcimonieux. Une façon de stocker un nombre exponentiel d'informations, pour un nombre de neurones fixé, est possible, mais sous certaines contraintes telles que les

classes de schéma à apprendre et la connectivité du réseau, par exemple le passage d'un réseau à un autre par matrice de permutation [59].

En vision par ordinateur, la mémoire associative peut être utilisée dans une tâche de classification par un apprentissage profond en l'ajoutant au modèle génératif DBN. En effet, une mémoire associative est utilisée après le passage de l'information à travers un empilement de réseaux de Boltzmann restreints afin d'associer les caractéristiques extraites avec l'étiquette d'une classe [61].

Contrairement au modèle de Hopfield, le CbNN ne dispose pas de fonction d'énergie. Par conséquent, ses applications sont limitées à la théorie avec des messages générés aléatoirement de manière uniforme. Cependant, les travaux récents consistent à traiter des données non-uniformes à l'aide de codages variés tels que celui de Huffman [19] ou par étiquetage [91]. Le CbNN peut être utilisé pour la classification en l'utilisant sous forme hétéro-associative. La première couche peut apprendre des caractéristiques (SIFT et GIST) extraites au préalable et la seconde couche apprend les étiquettes [49].

Une problématique inhérente aux mémoires associatives vient de leurs règles d'apprentissage dites instantanées qui gèrent mal les informations complexes (de dimension élevée) telles que des images. Par conséquent, les mémoires doivent passer par un pré-traitement tel qu'une extraction de caractéristiques. Les images ne sont pas des données indépendantes et identiquement distribuées (iid), et leurs corrélations accentuent donc la création de fausses mémoires.

## 2.6 Conclusion

Les réseaux de neurones sont désormais un outil indispensable dans les traitements de données. La révolution de ces réseaux provient de la possibilité d'utiliser une puissance de calcul devenue disponible via l'utilisation des cartes graphiques (GPU) pour accélérer les calculs. L'avènement de l'apprentissage profond faisant suite à l'utilisation de toujours plus de puissance de calcul pour améliorer les performances amène désormais l'optimisation des ressources à travers leur réduction pour être porté sur un grand panel de supports de calcul. Cette réduction des ressources passe par la simplification des réseaux existants ou un changement de paradigme. Les réseaux les plus performants s'appuient sur des architectures récurrentes utilisant de la mémoire. Afin de définir si l'utilisation de mémoire atténue le besoin de calcul, nous nous intéressons aux réseaux de neurones sous forme de mémoires associatives nécessitant pour fonctionner moins de ressources que les réseaux optimisés.

Les réseaux optimisés, qu'ils soient discriminatifs ou génératifs, prennent du temps à transformer les données en changeant leurs représentations afin de les différencier suivant leurs types, distributions, etc. L'apprentissage nécessite des ressources avec l'utilisation des méthodes de rétro-propagation ou de divergence contrastive. Cependant, une fois l'étape

d'apprentissage effectuée, la phase de test est rapide, puisque les données sont directement projetées dans l'espace créé à l'aide du produit matriciel.

En revanche, l'apprentissage par les mémoires associatives est instantané puisque l'information est directement encodée au sein du réseau. C'est l'étape de récupération qui prendra le temps de converger vers un objet préalablement appris ou une interpolation de ce dernier. Cependant, l'architecture simplifiée des mémoires associatives permet une simplification des calculs par rapport aux réseaux optimisés. Cette simplification peut aussi être une contrainte sur le réseau, ne lui permettant pas d'obtenir les mêmes performances qu'un réseau optimisé. En revanche, ces mémoires associatives sont adaptables à tout type de circuit, et leurs phases de récupération demandent moins de ressources que les réseaux optimisés.

Un critère de performance des mémoires associatives est leur capacité de stockage des informations apprises. Nous avons vu que, parmi les mémoires associatives les plus courantes, le CbNN se démarque de par sa parcimonie, qui lui permet une plus large diversité. Comme présenté dans le tableau 2.2, à nombre de neurones égal, le CbNN stocke davantage d'information avant qu'il ne soit incapable de récupérer une information sans ambiguïté. Par conséquent, en suivant ce critère primordial, le CbNN est la mémoire associative la plus performante [53]. En dépit de ses qualités, un problème inhérent au CbNN est que, contrairement à un modèle de Hopfield, il ne possède pas de fonction d'énergie permettant d'assurer presque sûrement une convergence vers un message appris par la mémoire. Ce défaut est dû au codage sous forme de cliques des messages, rendant difficile la correction par le réseau d'une information inconnue, et entraînant son problème de généralisation. Néanmoins, l'atout du CbNN est qu'il a été pensé sous forme d'architecture matérielle [76] afin d'être implanté dans des systèmes physiques sous forme de puces utilisant peu de ressources [20] [74].

Un dernier défi qui reste à lever pour le réseau CbNN est son utilisation sur des données réelles. En effet, celles-ci n'ont pas été étudiées minutieusement comme a pu l'être le réseau de Hopfield quand il représentait une solution alternative à des réseaux de neurones coûteux en calculs.

Nous présentons dans ce manuscrit plusieurs avancées relatives à l'utilisation du CbNN sur des données réelles, dans l'optique de répondre à la question suivante : pouvons-nous substituer le calcul par la mémoire ? La première ébauche de réponse à cette question passe par la compréhension du fonctionnement du réseau présentée dans le chapitre suivant.

# Chapitre 3

## Réseau de neurones à cliques

---

### Contenu

---

<b>3.1</b>	<b>Introduction</b> . . . . .	<b>36</b>
<b>3.2</b>	<b>Modèle de base</b> . . . . .	<b>37</b>
3.2.1	Architecture d'une mémoire . . . . .	37
3.2.2	Phase d'apprentissage . . . . .	38
3.2.3	Phase de récupération . . . . .	42
3.2.3.1	Règles dynamiques . . . . .	42
3.2.3.2	Règles d'activation . . . . .	44
<b>3.3</b>	<b>Analyse des défauts</b> . . . . .	<b>46</b>
3.3.1	Application à des données non uniformes . . . . .	47
3.3.2	Ambiguïtés . . . . .	47
3.3.3	Dimension . . . . .	50
3.3.4	Généralisation . . . . .	52
<b>3.4</b>	<b>Conclusion</b> . . . . .	<b>53</b>

---

## 3.1 Introduction

La télécommunication définit l'échange de signaux entre un point A et un point B. Les signaux peuvent correspondre à n'importe quel type de données comme du son ou de l'image, puisqu'ils sont toujours représentés sous forme de séquences binaires lors du transfert. Cette transmission peut se faire avec ou sans fil. Dans tous les cas, une télécommunication implique l'utilisation d'un émetteur et d'un receveur. Il est nécessaire que les données entre ces derniers transitent avec le moins de dégradation possible, voire aucune. En effet, le canal de transmission peut subir des interférences ajoutant alors du bruit au signal. Nonobstant le bruit, le théorème de Shannon [119] établit qu'il est possible de transmettre des informations presque sans dégradation mais, en contrepartie, le taux de transfert du canal doit être limité. Cette limite est posée de manière théorique. Afin de détecter et corriger les erreurs présentes dans les signaux, les canaux emploient des codages à leurs émissions qui seront décodés par le receveur. On parle de codes correcteurs d'erreurs.

Les premières méthodes de détection et correction s'appuient sur la redondance des signaux à transmettre, ou un ajout de données supplémentaires [56]. Désormais, les codes correcteurs les plus performants tels que les vérifications de parité à faible densité (ou *Low Density Parity Check*, LDPC) [46], ou les Turbo codes [15] tendent vers la limite théorique de Shannon pour le transfert et la correction du signal dans un canal [119], mais au prix d'un coût de calcul élevé. Cette complexité peut être réduite en modifiant la manière d'effectuer les calculs, par exemple en changeant le couple encodeur/décodeur.

Les méthodes initiales d'encodeur et décodeur s'appuient sur des méthodes d'algèbre linéaire et des statistiques calculées sur les données, elles sont donc sans mémoire. L'utilisation de cette dernière peut se faire à l'aide de la distance de Hamming [56] afin de retrouver, parmi un ensemble de messages décodés connus, celui qui a été encodé par la minimisation de cette distance ; c'est le principe du maximum de vraisemblance. Une autre forme de mémoire est définie par le réseau de Hopfield [63] qui repose sur une architecture de réseau de neurones récurrent. L'idée est d'apprendre, à travers l'utilisation de neurones, les messages que le receveur peut rencontrer, et ce avant la phase de décodage. Ainsi, la propagation dans le réseau sert de phase de correction. Par conséquent, si un message encodé est dégradé durant sa transmission, l'étape de décodage par le réseau de Hopfield corrige le message au moyen d'une convergence vers un message sans erreur appris au préalable. Le réseau de Hopfield a été largement utilisé comme correcteur de codes grâce à son architecture simple et la faisabilité de son implantation matérielle. Cependant, l'architecture du réseau limite sa diversité : le nombre de messages appris reste faible au regard du nombre de neurones disponibles.

Le réseau de neurones à base de cliques (CbNN, initialement GBNN pour *Gripon Berrou Neural Network*) reprend le concept du modèle de Hopfield, mais en ajoutant des améliora-

tions lui permettant d'être plus efficace dans le stockage de l'information [52], comme nous l'avons présenté dans le chapitre précédent.

Dans ce chapitre, nous présentons le modèle de réseau de neurones à cliques en détaillant et illustrant son architecture de type mémoire. En tant que telle, nous détaillons ses principes d'apprentissage à travers sa capacité de codage de l'information. L'important dans une mémoire associative étant la gestion de son espace de stockage, nous définissons la densité et expliquons comment la mesurer. Puisque nous souhaitons utiliser ce réseau pour reconstruire des données qui lui sont soumises, nous présentons également la phase de récupération. Celle-ci permet au réseau de reconstruire, à partir des informations apprises durant la phase d'apprentissage, une information dégradée qui lui est soumise. Après avoir décrit les qualités du réseau, nous concluons ce chapitre en présentant les limites du CbNN en soulignant les défauts que peut rencontrer le réseau avec d'autres types de données.

## 3.2 Modèle de base

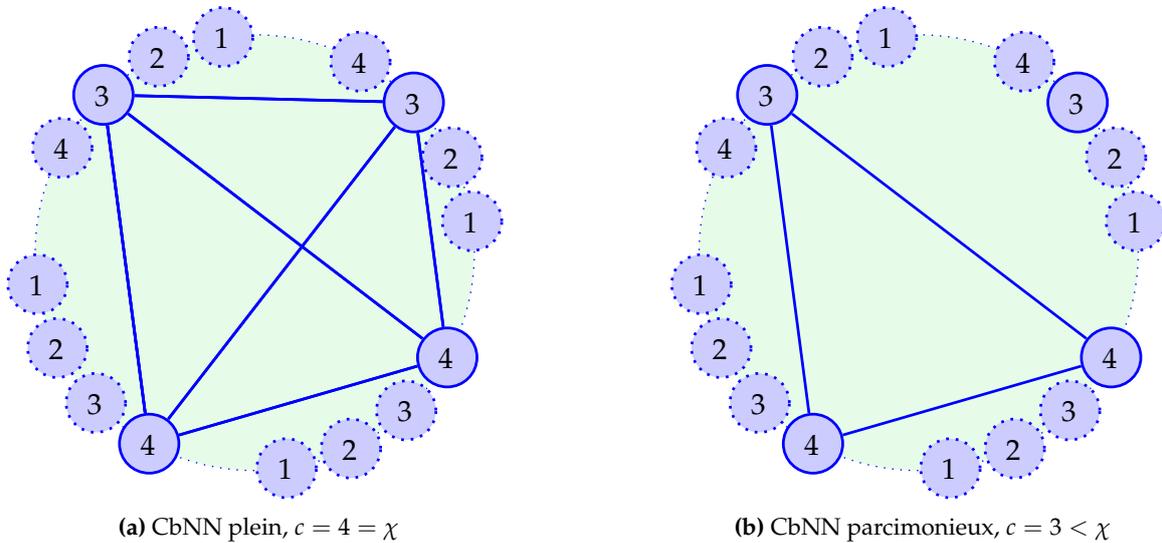
### 3.2.1 Architecture d'une mémoire

Le réseau de neurones à base de cliques (CbNN) est une mémoire auto-associative binaire réunissant  $\chi$  clusters possédant chacun  $l$  neurones. Le nombre de clusters dans le modèle initial [52] correspond à la partition des messages que l'on souhaite apprendre. Le nombre de neurones pour chaque cluster est égal aux possibilités prises par ces partitions de message. Par exemple, nous souhaitons apprendre le message binaire 10101111, son découpage en 4 parties de même dimension,  $\{10,10,11,11\}$ , entraîne la création d'un réseau composé de  $\chi = 4$  clusters. Chaque partie est un ensemble de 2 bits, le nombre de neurones par cluster est par conséquent  $l = 2^2 = 4$ . Le réseau possède un nombre total de neurones  $N = \chi \times l$ , soit 16 ici. L'apprentissage de ce message est illustré par la figure 3.1a.

On distingue deux structures de réseau qui dépendent des messages que l'on souhaite enregistrer dans la mémoire. En effet, cette différence de structure dépend du nombre  $c$  de clusters à activer pour représenter un message parmi les  $\chi$  clusters disponible dans le réseau. Dans le cas d'une utilisation de tous les clusters, le réseau sera dit plein, sinon il sera parcimonieux [4]. Ces deux réseaux sont illustrés respectivement par les figures 3.1a et 3.1b.

Comme le montre la figure 3.1, un message est représenté par un graphe  $G = (N, E)$  complet et non dirigé appelé *clique*, où les neurones caractérisant un message sont tous connectés les uns aux autres. La contrainte dite de parcimonie structurelle stipule qu'un seul neurone au maximum peut être activé dans un cluster. De cette contrainte découle la propriété qu'au sein d'un même cluster, aucun des neurones ne peut être connecté aux autres.

Pour un graphe avec  $N$  nœuds, soit  $A$  sa matrice booléenne d'adjacence associée. Sa dimension est  $N \times N$  et elle représente les connexions entre les nœuds. L'entrée  $A_{i,j}$  possède



**FIGURE 3.1** – Les deux structures possible de CbNN. Chaque réseau est composé de  $\chi = 4$  clusters comprenant chacun  $l = 4$  neurones. Le modèle plein utilise tous les clusters du réseau  $c = \chi$  pour représenter un message, tandis que le modèle parcimonieux utilise un sous-ensemble de clusters  $c < \chi$ .

la valeur 1 si les neurones  $i$  et  $j$  sont connectés. La matrice d'adjacence est symétrique puisque le graphe est non dirigé, induisant l'égalité entre les entrées  $A_{i,j}$  et  $A_{j,i}$ .

La matrice d'adjacence représente la mémoire du CbNN. Dans l'optique de conserver un réseau de neurones entièrement implantable sous forme de circuit logique [76], il est souhaité que la matrice d'adjacence conserve la binarité de ses entrées durant l'apprentissage, soit  $A_{i,j} \in \{0;1\}$ . Par conséquent, si une connexion entre deux neurones apparaît au travers de différents messages, la valeur dans la matrice d'adjacence restera "1", ce qui correspond à l'utilisation de la règle du seuillage (équation 2.37). Plus le réseau apprend de messages, et plus la matrice d'adjacence associée se remplit de "1" correspondant à toutes les connexions possibles entre les neurones tel qu'illustré par la distribution des activations de la figure 3.7. La matrice d'adjacence caractérise le CbNN. Dans la section suivante, nous expliquons comment cette matrice se remplit durant la phase d'apprentissage.

### 3.2.2 Phase d'apprentissage

La construction de l'architecture d'un CbNN dépend des messages à apprendre. Cependant, si un message n'est pas décomposable selon l'architecture du CbNN associé, le message ne pourra pas être appris. Chaque partie de ce message sera prise en compte au travers de l'activation d'un neurone dans le cluster associé. Les connexions entre les neurones activés forment la clique.

Un message est donc transformé sous forme de graphe représenté par une matrice d'adjacence. Par conséquent, la mémoire du réseau est l'union des matrices d'adjacence associées aux graphes appris. Soit  $W$  la mémoire du réseau, elle est définie telle que

$$W = \bigvee_i A_i. \quad (3.1)$$

Pour un message donné  $m_i \in \{0;1\}^{1 \times n}$  parmi l'ensemble  $\mathcal{M}$  soumis au réseau pour apprentissage, avec  $n$  le nombre de bits, soit  $T_i \in \{0;1\}^{1 \times N}$  les neurones activés pour représenter le message  $m_i$ . En reprenant l'exemple précédent avec l'apprentissage du message  $m_1 = 10 \mid 10 \mid 11 \mid 11$  décomposé en  $\chi = 4$  parties, son codage dans le réseau est

$$T_1 = 0010 \mid 0010 \mid 0001 \mid 0001, \quad (3.2)$$

avec un "1" pour le neurone activé dans chaque cluster correspondant à sa valeur binaire plus un, soit  $3 \mid 3 \mid 4 \mid 4$ , afin que chaque neurone représente une possibilité du codage binaire. Dans le cas d'un réseau parcimonieux, la valeur binaire n'est pas additionnée de un. Autrement dit, le codage est équivalent à la création d'un espace vectoriel constitué de vecteurs de norme 1. La matrice d'adjacence associée à  $m_i$  est la matrice carrée des activations des neurones pour représenter ce message. Elle s'écrit comme le produit de  $T_i$  et sa transposée

$$A_i = T_i^t T_i. \quad (3.3)$$

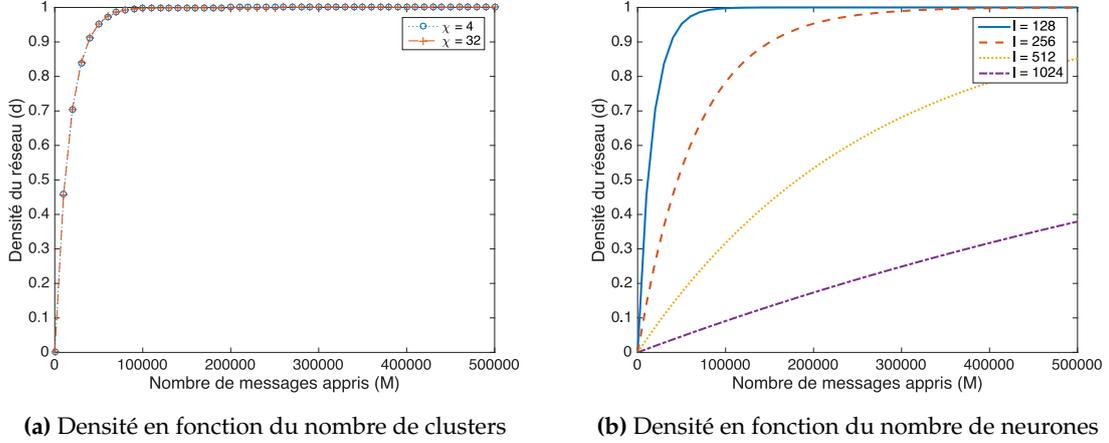
Une caractéristique importante d'une mémoire est le volume d'information qu'elle stocke, mesuré sous forme de densité. Le calcul est effectué en fonction de l'architecture du réseau, son nombre  $\chi$  de clusters, son nombre  $l$  de neurones, et sa structure (pleine ou parcimonieuse). Le nombre total de connexions possibles pour un graphe complet non dirigé avec  $N = \chi \times l$  nœuds est de  $(\chi l(\chi l - 1))/2$ , mais nous devons y retirer  $\chi(l(l - 1)/2)$  du fait de la contrainte de parcimonie structurelle, puisqu'aucun des neurones d'un même cluster ne peut être connecté aux autres. Le nombre de connexions possibles étant donnée une architecture de CbNN, formulé comme sa capacité  $Q$ , est défini comme

$$Q = \frac{\chi(\chi - 1)l^2}{2}. \quad (3.4)$$

Un message tiré suivant une distribution uniforme nécessite  $c(c - 1)/2$  connexions afin de construire une clique avec  $c$  clusters. La probabilité qu'une connexion particulière n'appartienne pas à cette clique est de

$$P_{out} = 1 - \frac{c(c - 1)}{\chi(\chi - 1)l^2}. \quad (3.5)$$

Après un stockage de  $M$  messages générés aléatoirement suivant une distribution uniforme, *c.-à-d.* identiquement et indépendamment distribuée, la probabilité qu'un message remplisse



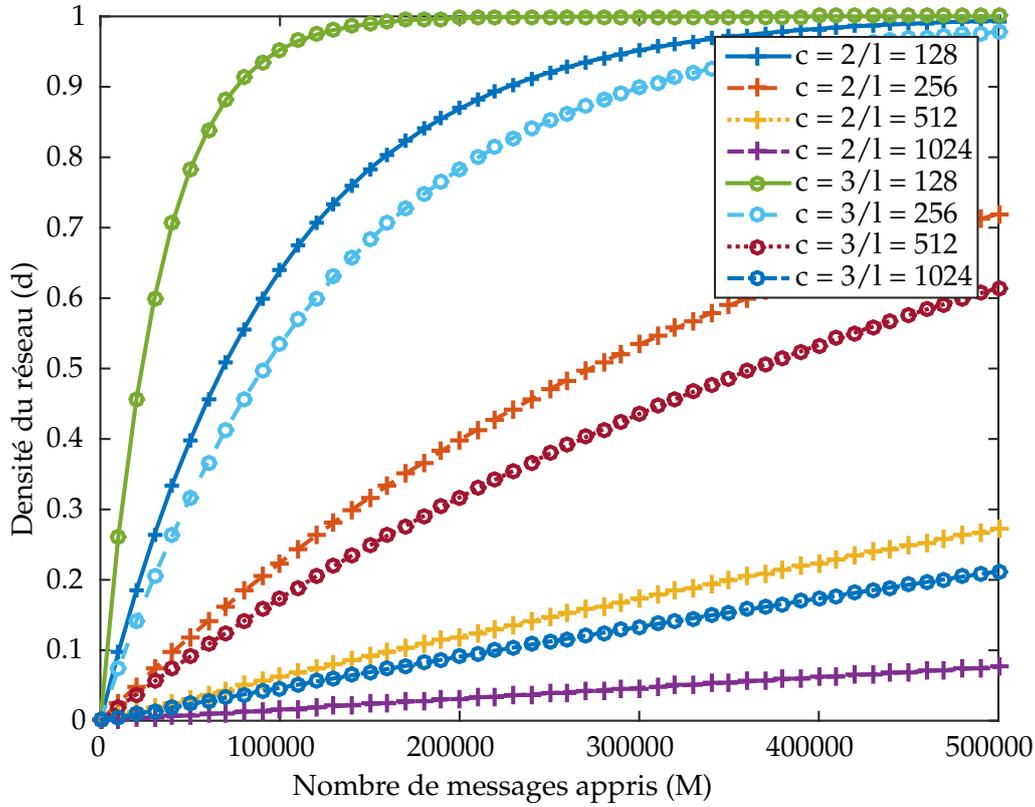
**FIGURE 3.2** – Évolution de la densité en fonction du nombre de clusters  $\chi$  et neurones  $l$  dans un réseau CbNN.

la mémoire, mesurée comme la densité, est donnée par [4] :

$$d = 1 - \left(1 - \frac{c(c-1)}{\chi(\chi-1)l^2}\right)^M, \quad (3.6)$$

avec  $c < \chi$  pour un réseau parcimonieux. Dans le cas d'un réseau plein, l'équation de la densité (3.6) devient,  $d = 1 - \left(1 - \frac{1}{l^2}\right)^M$  [52]. Cette dernière formulation implique que seule la variation des neurones dans un cluster est en mesure de faire évoluer différemment la densité (figure 3.2b). En effet, la figure 3.2a montre que, pour des réseaux CbNN apprenant des messages générés aléatoirement suivant une distribution uniforme, avoir un même nombre de neurones  $l = 128$ , mais différents nombres de clusters ( $\chi = 4$  et  $\chi = 32$ ) induit une évolution de densité identique. La figure 3.2b représente l'évolution de la densité de réseaux CbNN pleins avec un même nombre de clusters  $\chi = c = 4$  mais un nombre de neurones différents  $l = \{128, 256, 512, 1024\}$  associé à chaque réseau. Le nombre de messages uniformes qu'un réseau peut stocker dépend du nombre de neurones par cluster. Dans le cas d'un réseau parcimonieux, c'est le nombre de clusters actifs  $c$  qui sera le facteur limitant dans l'évolution de la densité comme illustré par la figure 3.3. Le but de la structure parcimonieuse est d'augmenter le nombre de messages stockables dans la mémoire. Ainsi, pour un même nombre de neurones, moins il y a de clusters actifs et plus le nombre de messages pouvant être stockés est important.

A partir du calcul de densité de l'équation (3.6), on peut dériver la diversité du réseau, qui s'exprime comme le nombre maximal de messages que peut contenir la mémoire étant donnée une architecture fixée. En premier lieu, l'efficacité  $\eta$  est calculée comme le ratio entre la quantité d'informations que possède le réseau après l'apprentissage de  $M$  messages, et sa



**FIGURE 3.3** – Évolution de la densité du réseau CbNN parcimonieux pour un apprentissage de messages générés aléatoirement suivant une distribution uniforme. La parcimonie est donnée par l'usage de clusters actifs  $c < \chi = 4$ .

capacité maximale  $Q$  [4] :

$$\eta = \frac{M \left( \log_2 \left( \binom{c}{\chi} \right) + c \log_2(l) \right)}{Q} = \frac{2M \left( \log_2 \left( \binom{c}{\chi} \right) + c \log_2(l) \right)}{\chi(\chi - 1)l^2}. \quad (3.7)$$

Ainsi, pour une efficacité égale à 1, on obtient la borne maximale de messages  $M_{max}$  que peut apprendre le réseau, soit

$$M_{max} = \frac{\chi(\chi - 1)l^2}{2 \left( \log_2 \left( \binom{c}{\chi} \right) + c \log_2(l) \right)}. \quad (3.8)$$

L'apprentissage s'effectue en encodant les messages soumis au réseau afin de remplir la matrice d'adjacence  $W$  du réseau. La densité d'information de cette matrice permet de définir un apprentissage optimal, pour lequel il est préférable d'avoir une structure parcimonieuse avec peu de clusters actifs, mais possédant de nombreux neurones. Cette information permet aussi de calculer le nombre maximal de messages qu'un CbNN peut apprendre en

fonction de son architecture. Après avoir décrit la phase d'apprentissage de messages, nous allons présenter la phase de récupération de messages partiellement dégradés.

### 3.2.3 Phase de récupération

Le CbNN est une mémoire associative car il corrige les données soumises au réseau en utilisant les informations stockées dans sa mémoire. Le mécanisme de récupération est réalisé par l'itération de deux processus successifs appelés règles dynamique et d'activation. L'algorithme 2 détaille l'itération des deux processus, et la figure 3.4 les illustre.

#### 3.2.3.1 Règles dynamiques

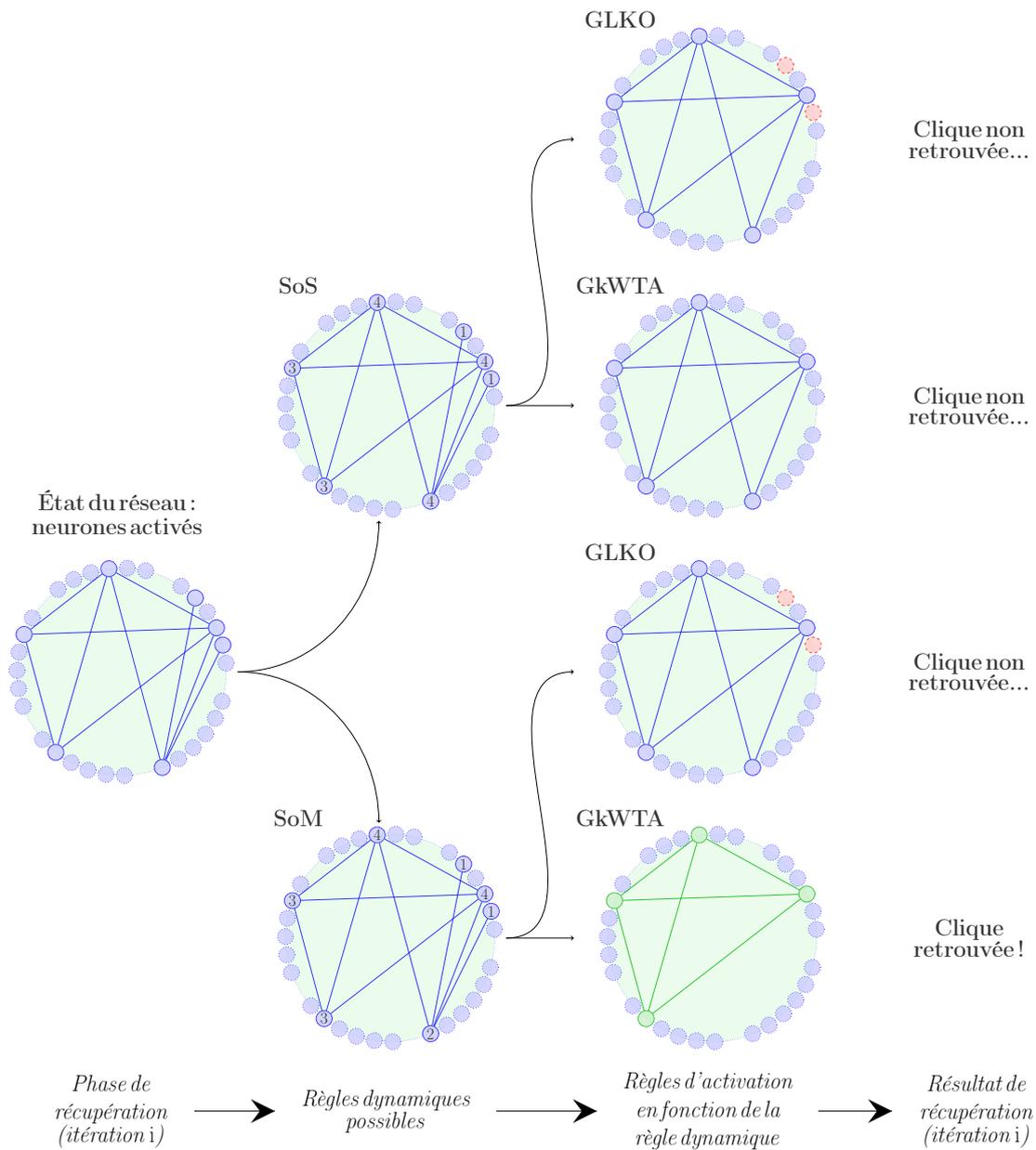
Une règle dynamique met en avant les neurones qui sont les plus à même d'avoir un lien avec l'entrée soumise pour correction. Ce lien est mesuré par un score calculé à partir du nombre de neurones en commun en utilisant la matrice d'adjacence. Le score d'un neurone peut être calculé de diverses manières. Soit  $v_{ij}$  le  $j^{\text{ème}}$  neurone du cluster  $i$ , la méthode classique pour calculer son score  $\lambda_{ij}$  est de sommer ses connexions aux autres neurones  $v_{i'j'}$  à travers la matrice d'adjacence  $W$ , soit

$$\lambda_{ij} = \gamma v_{ij} + \sum_{i'=1}^{\chi} \sum_{j'=1}^l W_{(ij)(i'j')} v_{i'j'}. \quad (3.9)$$

Le paramètre prédéfini  $\gamma$  est l'effet mémoire correspondant à l'impact voulu du neurone dont on calcule le score, il est généralement égal à 1. Cette règle dynamique est appelée *Sum-of-Sum* (SoS). Elle se base sur la fréquence d'usage des connexions d'un neurone, ce qui peut apporter un biais. En effet, un neurone peut avoir un score élevé uniquement parce qu'il possède davantage de connexions qu'un autre neurone, au détriment de son appartenance réelle à la clique recherchée. Autrement dit, son score est calculé seulement parce qu'il est connecté à un neurone commun à l'entrée. Pour remédier à ce biais, au lieu de prendre en compte toutes les connexions entrantes du neurone  $v_{ij}$ , il est possible de maximiser le nombre de clusters qui lui sont connectés. Cette règle dynamique est appelée *Sum-of-Max* (SoM),

$$\lambda_{ij} = \gamma v_{ij} + \sum_{i'=1}^{\chi} \max_{1 \leq j' \leq l} (W_{(ij)(i'j')} v_{i'j'}). \quad (3.10)$$

La correction d'une entrée doit se terminer si possible par la reconstruction d'une clique apprise au préalable. De par la contrainte de parcimonie structurelle du réseau, la correction implique l'activation d'au plus un neurone par cluster. La sélection de ces neurones s'effectue par une règle d'activation.



**FIGURE 3.4** – Déroulement de la phase de récupération à l'itération  $i$  avec les différentes règles dynamiques et règles d'activation. Pour cet exemple  $k = 4$ .

### 3.2.3.2 Règles d'activation

La sélection des neurones restant activés à l'itération suivante s'appuie sur les scores calculés par la règle dynamique. Nous rappelons qu'au plus un neurone par cluster doit être activé. Par conséquent, la règle d'activation la plus simple consiste à sélectionner pour chaque cluster le neurone possédant le score le plus élevé. C'est le *Winner take all* (WTA). Pour chaque cluster  $i$ , le score maximal est défini par  $\theta_i$  tel que

$$v_{ij} = \begin{cases} 1 & \text{si } \lambda_{ij} = \theta_i, \\ 0 & \text{sinon.} \end{cases} \quad (3.11)$$

L'utilisation du WTA peut entraîner l'activation de plusieurs neurones au sein d'un même cluster si leurs scores sont égaux, induisant alors un problème d'ambiguïté. Cette problématique sera discutée plus en détail lors de l'analyse des défauts du CbNN.

Le WTA ne peut pas être utilisé dans un réseau possédant une structure parcimonieuse. En effet, pour un cluster  $i$  non actif, son score maximal est  $\theta_i = 0$  impliquant l'activation d'un neurone qui n'a pas lieu d'être. Par conséquent, les clusters actifs dépendent des messages et sont donc aléatoires. La règle d'activation pour un réseau parcimonieux est donc modifiée en définissant  $\theta$  non pas pour chaque cluster, mais pour l'ensemble du réseau, conduisant alors à la règle du nom de *Global Winner Take All* (GWTA). Afin de ne pas perdre de l'information en restreignant  $\theta$  au score maximal du réseau, on pose  $\theta$  comme étant la valeur minimale des  $\alpha$  plus hauts scores du réseau. Le paramètre  $\alpha$  est un entier fixé par l'utilisateur pour définir le nombre de scores sélectionnés afin d'accroître la robustesse de la règle d'activation. En effet, prendre le meilleur du réseau risquerait de limiter la reconstruction d'une clique du fait du faible nombre de neurones activés. La sélection s'effectue suivant l'équation (3.11) mais le seuil devient fixe sur tout le réseau, impliquant que  $\theta_i$  devienne  $\theta_\alpha$  :

$$v_{ij} = \begin{cases} 1 & \text{si } \lambda_{ij} \geq \theta_\alpha, \\ 0 & \text{sinon.} \end{cases} \quad (3.12)$$

Une variante du GWTA est le GkWTA avec  $k$  représentant une valeur optimale de  $\alpha$  et étant défini comme le nombre de clusters actifs  $c$  autorisés pour exprimer le message dans le réseau.

Le problème rencontré par l'utilisation des règles basées sur le WTA consiste en la restriction appliquée aux neurones ayant les scores les plus élevés, revenant ainsi à effectuer une sélection brutale. Au contraire, il est possible d'éliminer les neurones ayant les scores les plus faibles, c'est le *Loser Kicked Out* (LKO). Pour la même raison que le WTA, le LKO n'est utilisable que dans un réseau plein. Son adaptation pour un CbNN de structure parcimonieuse est le GLKO. Le seuil  $\theta$  est défini comme étant le maximum des  $\beta$  (paramètre entier

fixé comme  $\alpha$ ) plus bas scores,

$$v_{ij} = \begin{cases} 1, & \text{si } \lambda_{ij} > \theta_{\beta} \\ 0, & \text{sinon.} \end{cases} \quad (3.13)$$

Une dérivée du GLKO est le GkLKO qui reprend le même principe que le GkWTA. La correction n'est pas limitée à l'usage d'un seul type de règle d'activation. Une stratégie consiste à affiner l'entrée en fixant un seuil peu restrictif avec un GWTA, puis à désactiver graduellement les neurones restants avec un GLKO. Par conséquent, la correction est découpée en deux phases présentées par l'algorithme 2. Ainsi, pour une utilisation optimale du GLKO, il est conseillé de commencer en phase 1 par un GWTA, puis de continuer la phase 2 avec le GLKO.

---

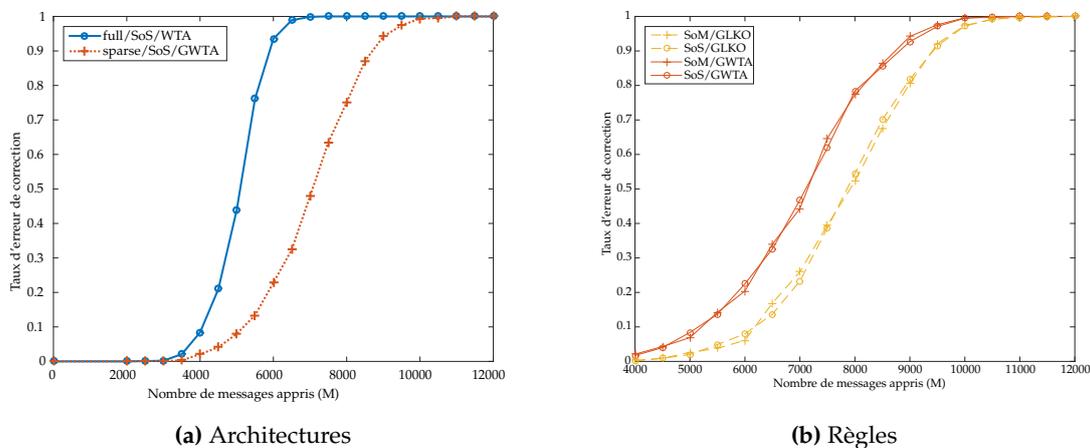
**Algorithme 2** Processus de récupération [2]

---

- 1: *Insertion d'un message*
  - 2: *Application d'une règle dynamique*
  - 3: **Phase 1 :**
  - 4:     *Application d'une règle d'activation*
  - 5:     *Application d'une règle dynamique*
  - 6: **Phase 2 :**
  - 7:     **Tant que** critère d'arrêt non atteint **faire**
  - 8:         *Application d'une règle d'activation*
  - 9:         *Application d'une règle dynamique*
  - 10:     **fin tant que**
  - 11: sortie  $\leftarrow$  neurones actifs
- 

La structure d'un réseau (pleine ou parcimonieuse) influe directement sur sa capacité à stocker plus de messages du fait d'un nombre inférieur de clusters actifs pour représenter un message. Par conséquent, l'erreur de reconstruction est améliorée pour un réseau parcimonieux (figure 3.5a).

Les performances de récupération dépendent des règles utilisées comme l'illustre la figure 3.5b. La différence entre les règles d'activation GWTA et GLKO s'explique par le seuillage effectué. En effet, tandis qu'avec le GWTA les neurones activés sont ceux possédant les scores les plus élevés, le GLKO désactive ceux avec les plus mauvais scores. Par conséquent, l'usage du GWTA entraîne une perte d'information due aux neurones désactivés par leurs mauvais scores en dépit d'une probabilité non nulle d'appartenir à la clique recherchée. Cependant l'usage du GLKO nécessite un nombre d'itérations supérieur au GWTA. La figure 3.5b montre que sur des données uniformes et une architecture parcimonieuse, la différence des scores entre les méthodes SoS et SoM est minime. En effet, la combinaison de données uniformes et d'une architecture parcimonieuse entraîne l'activation de neurones dans des clusters distincts, limitant ainsi pour un même cluster l'activation de plusieurs neurones. Pour rappel, la règle SoM compte, pour un neurone donné, le nombre de clusters



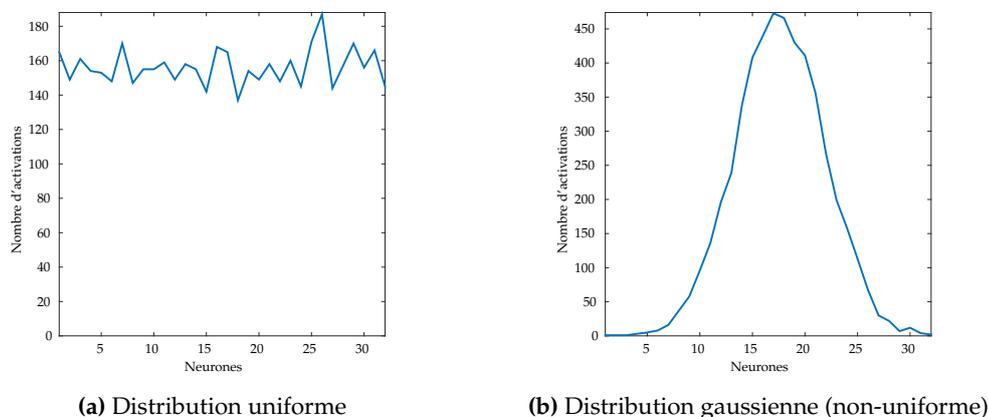
**FIGURE 3.5** – Taux d'erreur de correction entre des réseaux d'architectures différentes (a), et l'utilisation des diverses règles durant la phase de récupération (b), en fonction du nombre de messages uniformes appris. Les deux réseaux ont le même nombre de clusters  $\chi = 20$  comportant  $l = 64$  neurones.  $c = \chi$  pour le réseau plein et  $c = 12$  pour le réseau parcimonieux. Le taux d'erreur provient de la phase de récupération sur 1 000 tests d'effacement concernant 25% des clusters actifs de chaque réseau.

auxquels il est connecté, tandis que le SoS compte toutes les connexions.

Le processus de correction est itératif, et sa convergence est définie par un critère d'arrêt. Nous rappelons les quatre critères définis dans [2]. Ainsi, la correction peut s'arrêter quand : (1) le nombre d'itérations maximum fixé est atteint ; (2) les mêmes neurones sont activés après deux itérations successives ; (3) tous les scores des neurones activés sont égaux ; (4) une clique a été trouvée, ou autrement dit  $c$  neurones sont actifs et connectés. Nous pourrions assimiler le quatrième critère d'arrêt au troisième, puisque si une clique est trouvée, les scores des neurones activés doivent être égaux. Cependant, comme nous allons le présenter dans la section suivante, plusieurs neurones activés peuvent avoir le même score sans représenter une unique clique. Ce défaut inhérent au type des données apprises correspond à une ambiguïté.

### 3.3 Analyse des défauts

Dans cette section, nous décrivons les expérimentations menées sur le réseau afin de déterminer ses limites et en analyser ses défauts. La validation initiale du modèle s'appuie sur des données générées artificiellement, suivant une distribution uniforme, soit identiquement et indépendamment distribuée. Nous nous intéressons ici au comportement du réseau dans le cas plus réaliste de données non uniformes.



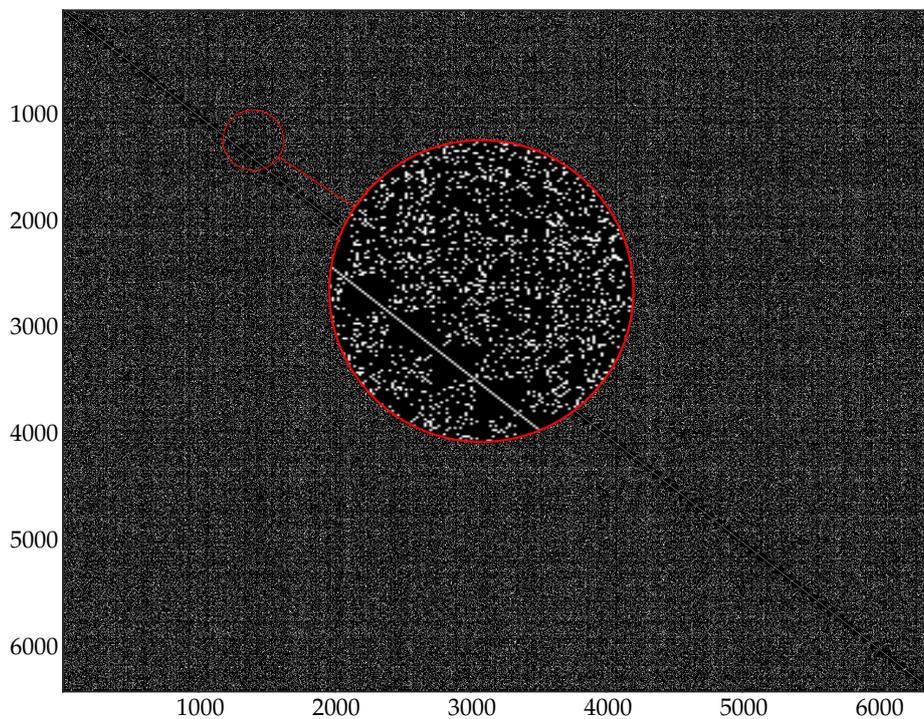
**FIGURE 3.6** – Distribution de l'activité neuronale sur 1 000 messages en fonction de distributions différentes apprises par un CbNN composé de  $\chi = 5$  clusters renfermant chacun  $l = 32$  neurones.

### 3.3.1 Application à des données non uniformes

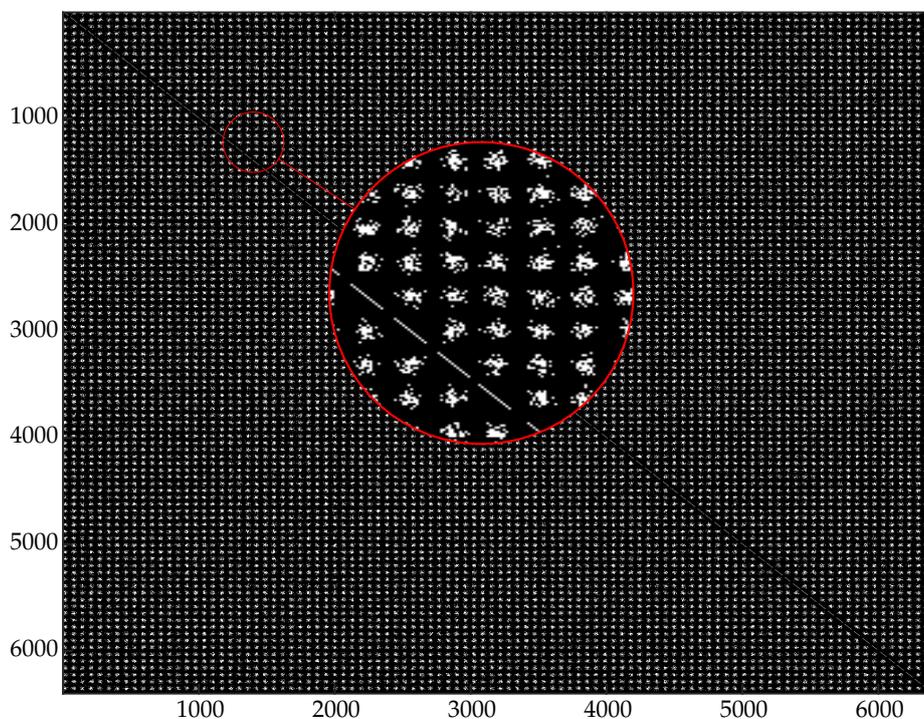
L'apprentissage de données aléatoires (figure 3.6) entraîne une répartition des activations correspondant à la distribution utilisée dans la génération aléatoire des messages (figure 3.7). Les données tirées selon une distribution non-uniformes, *c.-à-d.* gaussienne dans ce cadre d'expérimentation, amènent la sollicitation de certains neurones plus que d'autres (figure 3.7b). Par conséquent, pour un même nombre de messages appris, la densité est plus faible (figure 3.8a) qu'avec des données uniformes. La différence de densité conduit à une erreur de récupération plus élevée puisque moins de neurones contiennent les informations apprises (figure 3.8b). En conclusion, le réseau n'est pas en mesure de sélectionner correctement les neurones pour représenter l'information soumise à récupération, c'est le phénomène d'ambiguïté. Nous pouvons déterminer de manière empirique (approximativement à l'aide de la figure 3.8) la limite de densité afin de maintenir une correction optimale. Dans le cas d'une distribution uniforme, la limite est de  $d = 0,8$ . Elle est inférieure à  $d = 0,2$  pour des données suivant une distribution gaussienne.

### 3.3.2 Ambiguïtés

Une ambiguïté apparaît quand le réseau n'est pas en mesure de décider l'activation d'un neurone de façon univoque. En effet, durant la phase de récupération, la règle dynamique peut donner un score identique à deux neurones issus du même cluster, d'où la subtilité entre les troisième et quatrième critères d'arrêt. Cependant, la contrainte de parcimonie structurelle oblige à ce que seul un neurone soit activé par cluster. Par conséquent, la règle d'activation opérant dans un même cluster, plusieurs neurones aux scores identiques

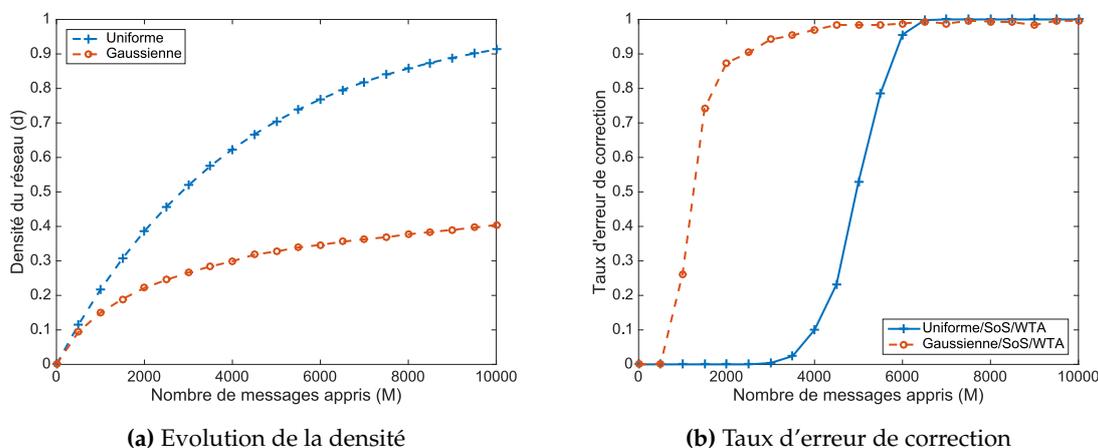


(a) Répartition suivant une distribution uniforme



(b) Répartition suivant une distribution gaussienne (non uniforme)

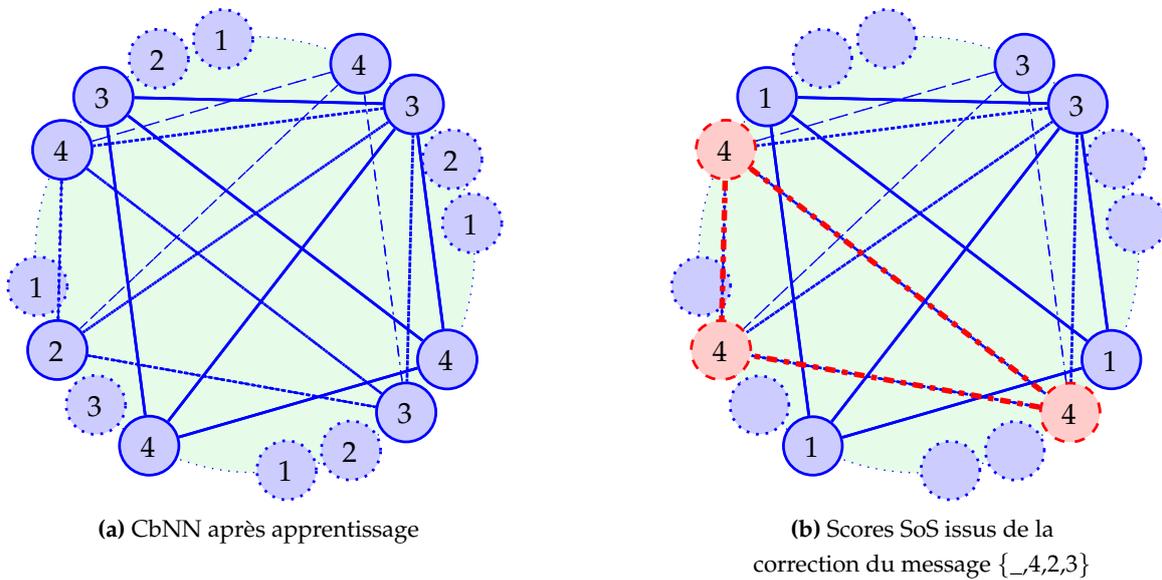
**FIGURE 3.7** – Répartition dans la matrice d'adjacence de 1 900 messages appris par un CbNN de structure pleine composé de  $\chi = c = 100$  clusters renfermant chacun  $l = 64$  neurones. La contrainte structurelle est visible sur la diagonale des matrices.



**FIGURE 3.8** – Densité et taux d'erreur de correction en fonction du nombre de messages appris. Les deux réseaux ont la même structure pleine avec  $\chi = c = 20$  clusters comportant chacun  $l = 64$  neurones. L'un apprend des messages suivant une distribution uniforme, l'autre gaussienne. La correction est effectuée à l'aide de la règle dynamique SoS et la règle d'activation WTA sur un ensemble de 1 000 messages de test avec 25% clusters effacés.

seront activés conduisant à l'apparition d'un chevauchement de cliques. En effet, les neurones sélectionnés peuvent appartenir à des cliques différentes. Ces neurones reconstruiront chacun leur propre clique durant la phase de correction. Par conséquent, le réseau produit diverses parties de cliques (initialement indépendantes) au lieu d'une seule clique. De plus, si la contrainte structurelle n'est pas respectée, le décodage des cliques n'est pas possible, sinon une partition d'un message se retrouvera avec deux valeurs différentes (le CbNN n'est pas un réseau quantique). Le problème consiste à définir le neurone qui devra être activé.

Plus la densité est élevée et plus grande est la probabilité de rencontrer une ambiguïté, et ce d'autant plus si les messages à apprendre sont non-uniformes. En effet, dans ce cas, l'apparition des ambiguïtés est amplifiée par la répartition des activations. La figure 3.7b illustre les blocs de connexions qui, du fait de la distribution non-uniforme, laissent le même ensemble de neurones activés. Par la suite, ces derniers représenteront la majorité des messages appris. La phase de récupération n'est alors plus en mesure de dissocier explicitement les cliques, entraînant ainsi l'apparition d'ambiguïtés. L'exemple de la figure 3.9 illustre la problématique d'ambiguïté rencontrée en présence de scores identiques dans le premier cluster, après l'apprentissage de trois messages uniformément générés. Le résultat souhaité est celui en tirets, mais le réseau produira le résultat indiqué en pointillé. Dans cet exemple particulier, les deux résultats sont satisfaisants, mais avec un réseau composé de centaines de clusters et de milliers de neurones, la combinaison de ces conditions conduit à la production d'un message non souhaité, et par conséquent à un échec de la correction.



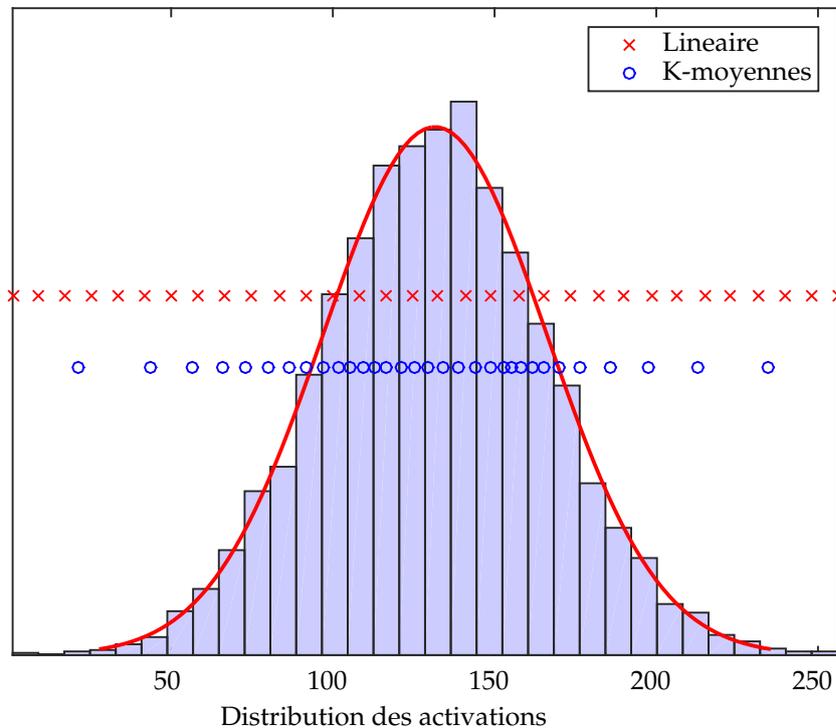
**FIGURE 3.9** – Illustration de l'ambiguïté dans le cas de la correction d'un message avec un CbNN composé de  $\chi = 4$  clusters possédant chacun  $l = 4$  neurones. Les messages appris sont  $\{3,3,4,4\}$  (solide),  $\{4,4,2,3\}$  (tirets) et  $\{3,4,2,3\}$  (pointillés). La règle dynamique de la phase de correction est le SoS.

Durant ces expérimentations, les données ont été générées aléatoirement. Nous verrons dans la prochaine section l'utilisation du CbNN avec des données réelles.

### 3.3.3 Dimension

Les images naturelles sont des collections de pixels corrélés rendant dès lors les données non-uniformes. Par conséquent, les problématiques précédentes surviennent lors de l'utilisation d'un CbNN sur des images. En outre, l'application du CbNN à des images amène un nouveau défi qui est relatif au problème de dimension.

La figure 3.1 induit une représentation spatiale qui peut être utilisée pour des images qui sont d'une dimension supérieure aux données initialement utilisées par le CbNN. Dans le cas de l'apprentissage d'images basé pixels, une approche naïve consiste à fixer le nombre de clusters égal au nombre de pixels, et le nombre de neurones pour chaque cluster à la profondeur du pixel. Si les images traitées sont de taille  $256 \times 256$  pixels et chacun d'eux est codé en couleurs avec 24 bits, le nombre de neurones nécessaires est de  $256^2 \times 2^{24} = 1,0995 \times 10^{12}$ . Par conséquent, la matrice d'adjacence du réseau associée aux messages à apprendre posséderait un nombre total de connexions possibles égal à  $9.1873 \times 10^{18}$  en prenant en compte la parcimonie structurelle (équation 3.4). Un codage de 1 bit par entrée de la matrice d'adjacence entraîne un besoin de  $1,15 \times 10^9$  Go de mémoire, ce qui n'est pas commun sur les ordi-



**FIGURE 3.10** – Position des moyennes mobiles sur des données non-uniforme en fonction des méthodes de quantification linéaire et K-moyennes.

nateurs. Par conséquent, les dimensions des images (résolution et profondeur), doivent être modifiées afin de permettre l'apprentissage d'images par le réseau. Les clusters dépendent de la résolution mais, comme illustré durant la phase d'apprentissage, ce n'est pas le nombre de clusters qui est important, mais celui des neurones. Le changement de profondeur réalisé par une quantification est de ce fait primordial.

Un choix de quantification consiste à utiliser la méthode des K-moyennes, avec  $K$  le nombre de neurones souhaités par cluster. Ainsi, le neurone à activer dans un cluster associé à un pixel est relatif à la proximité de sa position à une valeur obtenue à l'aide des K-moyennes. La figure 3.10 illustre notre choix de l'utilisation de la quantification à l'aide de la méthode des K-moyennes (le choix de la ligne de placement des méthodes est arbitraire). En effet, sur des données non-uniformes, la méthode linéaire nécessite l'utilisation d'un sous-ensemble de neurones pour représenter la majorité de l'information. En revanche, la méthode des K-moyennes adapte les moyennes mobiles à la distribution, permettant de dissocier plus efficacement les neurones à activer.

Cependant, l'utilisation de la méthode des K-moyennes apporte un biais par rapport à l'information à apprendre du fait de l'approximation de l'ensemble des images par les

moyennes mobiles générées. En effet, d'une part l'initialisation des moyennes mobiles est aléatoire, et d'autre part celles-ci évoluent différemment en fonction du nombre d'images à apprendre. Par exemple, si nous souhaitons apprendre 100 images, la méthode des K-moyennes sera appliquée à ces images, mais si on souhaite en apprendre davantage ultérieurement (apprentissage incrémental), les images suivantes dépendront des moyennes mobiles générées sur les 100 premières images induisant une perte d'information. Par conséquent, dans l'optique d'obtenir des résultats optimaux, le premier apprentissage doit être réalisé sur une large sélection d'images. Cependant, les travaux dans cette thèse peuvent aussi être appliqués à d'autres stratégies de quantification, comme la quantification vectorielle.

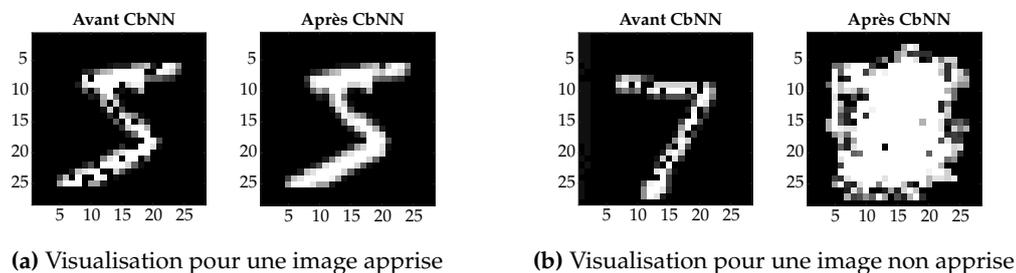
D'autres méthodes de réduction de dimension peuvent être utilisées pour apprendre des images, comme les descripteurs locaux (SIFT par exemple). Cependant, le CbNN est un réseau s'appuyant sur la reconstruction d'information. Par conséquent, l'usage de méthodes réduisant les images à leurs descripteurs rend caduque la possibilité d'utiliser le CbNN dans le cas de la reconstruction d'images.

Afin de valider le modèle, les expérimentations ont porté sur des messages générés artificiellement. De plus, les performances de correction s'appuient seulement sur des messages préalablement appris mais dégradés. Dans la prochaine section, nous décrivons la correction du CbNN en présence de données inconnues au réseau.

### 3.3.4 Généralisation

Le CbNN est capable de reconstruire uniquement des messages appris au préalable comme l'exemple de la figure 3.11a. La correction de données inconnues ne fonctionne pas comme le montre la figure 3.11b. En effet, la convergence vers une clique est accomplie si et seulement si le message soumis possède une partie commune exclusive à un message appris au préalable. Cependant, si ce n'est qu'une faible partie, la probabilité de retrouver la clique apprise restera faible. Si rien n'est connu du message soumis à correction, le CbNN ne sera pas en mesure de converger vers une unique clique apprise.

Cette problématique s'explique par le codage des messages sous forme de cliques indépendantes les unes des autres pendant la phase d'apprentissage. Bien que des connexions soient partagées entre les cliques, il n'existe pas de corrélation entre elles si elles sont générées à partir de données uniformes, étant indépendantes les unes des autres. Par conséquent, les neurones représentant des cliques différentes sont mis en valeur, conduisant à la production d'une composée de parties de cliques apprises par le réseau. La visualisation de la généralisation sur des séquences aléatoires n'est pas judicieuse, mais l'utilisation d'images révèle le problème auquel nous sommes confrontés. Comme l'illustre la figure 3.11a, la récupération des clusters effacés est un succès car le reste du message avait été appris. Toutefois, à la soumission d'une image inconnue du réseau, le résultat de la correction est l'union de



**FIGURE 3.11** – Visualisation de la correction par le CbNN d’une image apprise et d’une image inconnue. L’architecture parcimonieuse du réseau est de  $\chi = 784$  clusters comportant  $l = 16$  neurones chacun et  $c = 166$ . 50 images ont été apprises portant la densité à  $d = 0,0064312$ . Pour la correction, 30 clusters actifs ont été effacés pour les deux exemples.

cliques différentes. L’image 3.11b est le résultat de l’activation de plus d’un neurone par cluster du fait de l’échec de la correction suite aux ambiguïtés. Pour la visualisation, nous avons effectué un filtrage à l’aide d’un WTA.

## 3.4 Conclusion

Le CbNN est un réseau de neurones caractérisé par une matrice d’adjacence correspondant à la mémoire où l’information est encodée, et qui réalise des calculs afin d’extraire l’information. La particularité du CbNN vient de sa façon de stocker l’information et son efficacité à retrouver celle-ci. La première caractéristique de cette mémoire est mesurée sous forme de densité, qui permet de calculer l’espace de stockage. Cet espace prend la forme d’une matrice d’adjacence qui recense les connexions entre les neurones. Ces derniers sont regroupés sous forme de clusters, qui par leurs liaisons forment un graphe complet nommé clique représentant l’information encodée dans le réseau. La seconde caractéristique est mesurée en testant la phase de récupération sur des données apprises au préalable, puisque sinon le réseau ne serait pas en mesure de converger vers une clique apprise. La phase de récupération s’effectue par des itérations alternant une règle dynamique qui calcule les scores des neurones et une règle d’activation qui sélectionne les neurones devant rester activés en fonction de critères définis par la règle utilisée.

L’efficacité de correction, sur des messages générés aléatoirement suivant une distribution uniforme, décroît quand le réseau encode des messages ayant une distribution non-uniforme. La décroissance de l’efficacité est due à la répartition des entrées dans la mémoire qui suit la distribution des données apprises. Cette distribution non-uniforme entraîne des ambiguïtés durant la phase de récupération qui correspondent à des chevauchements

de cliques, d'où la réduction d'efficacité de la correction sur ce type de données. Dans le contexte de l'analyse d'image, la structure du réseau devient son point faible du fait de la dimension des données à apprendre. Nous présenterons dans le chapitre suivant nos contributions afin de résoudre les différentes problématiques inhérentes au CbNN que nous avons identifiées jusqu'ici.

# Chapitre 4

## Contributions au modèle original

---

### Contenu

---

<b>4.1</b>	<b>Introduction</b>	<b>56</b>
<b>4.2</b>	<b>Gestion de l'ambiguïté</b>	<b>57</b>
4.2.1	Solutions de la littérature	57
4.2.2	Modèle à couches multiples	58
4.2.2.1	Un réseau à double couche	58
4.2.2.2	Extension à N couches	64
4.2.2.3	Conclusion	68
4.2.3	Pénalisation	69
4.2.3.1	Expérimentations sur des données artificielles	72
4.2.3.2	Expérimentations sur des images	74
4.2.3.3	Conclusion	75
<b>4.3</b>	<b>Classification</b>	<b>76</b>
4.3.1	Méthode	78
4.3.2	Expérimentations	82
<b>4.4</b>	<b>Généralisation par a priori</b>	<b>85</b>
4.4.1	Utilisation d'a priori	85
4.4.2	Expérimentations	86
<b>4.5</b>	<b>Conclusion</b>	<b>89</b>

---

## 4.1 Introduction

L'usage actuel du réseau à cliques (CbNN) est restreint à la démonstration de performances en terme de récupération d'informations s'appuyant sur la théorie des codes correcteurs. Notre objectif est d'utiliser le CbNN dans le contexte de la vision par ordinateur, où les données ne sont plus uniformes. Bien que la littérature se soit récemment concentrée sur l'usage du CbNN avec des données de type non-uniformes, celles-ci restent toujours générées aléatoirement.

A travers la présentation des mécaniques fondamentales du CbNN dans le chapitre précédent, nous avons soulevé les problématiques inhérentes à l'utilisation de données non-uniformes avec le réseau. En effet, l'apprentissage de données non-uniformes entraîne une répartition moins homogène des informations au sein du réseau qu'avec des données uniformes. Cette non homogénéité est due à la sauvegarde de l'information par un sous-ensemble de neurones, ce qui devient le facteur limitant du réseau en terme de récupération. Lors de la phase de récupération, le réseau est incapable de dissocier les neurones du sous-ensemble actif afin de représenter l'information recherchée. L'incapacité de dissociation induit une ambiguïté. Les contributions récentes se concentrent sur la gestion des ambiguïtés par le réseau afin d'atteindre les performances affichées par le CbNN avec des données uniformes. Toutefois, les données non-uniformes restent générées aléatoirement et n'illustrent pas la capacité du réseau à traiter des problèmes réalistes, comme ceux rencontrés en vision par ordinateur.

Nous souhaitons utiliser le pouvoir correcteur du CbNN sur des images, avec pour objectif une tâche de classification des données reconstruites par le réseau. Autrement dit, nous désirons utiliser la mémoire associative du réseau afin de classer des données qui n'ont pas été apprises par le réseau, ce qui nous amène au problème de généralisation discuté dans le chapitre précédent. Initialement, le CbNN est incapable de récupérer une information issue de sa mémoire sur la base de données soumises inconnues, du fait de l'impossibilité de dissocier les informations apprises. Par conséquent, l'utilisation d'images est une excellente motivation pour améliorer le réseau à cliques dans sa gestion des données non-uniformes et sa capacité à généraliser l'information.

Ce chapitre présente les contributions que nous avons apportées au modèle initial du CbNN pour en atténuer les défauts. La première partie concerne la gestion de l'ambiguïté, et présente à la fois les contributions de la littérature et les nôtres. Ces dernières consistent notamment en un changement d'architecture, en étendant le CbNN en un modèle composé de plusieurs couches offrant l'avantage d'une plasticité et une meilleure gestion de la densité. Outre le changement d'architecture, nous présentons aussi un complément aux règles de récupération avec l'usage d'une pénalisation [69] visant à améliorer la dissociation des cliques apprises dans la mémoire. La seconde partie du chapitre s'intéresse à la tâche de clas-

sification d’images en utilisant les principes du CbNN. En effet, nous intégrons l’étiquette (ou la classe des images) dans le codage des images. La classification est réalisée en exploitant la capacité correctrice du réseau. La dernière partie concerne la possibilité qu’a le réseau de généraliser l’information. Nous discutons d’un changement au sein des règles de récupération à l’instar de la méthode de pénalisation, mais ici sous forme d’a priori permettant d’avantager les neurones actifs du réseau qui sont corrélés à une entrée inconnue.

## 4.2 Gestion de l’ambiguïté

Comme discuté dans le chapitre précédent, l’ambiguïté a une probabilité accrue d’apparaître en présence de données non-uniformes. En effet, l’apprentissage de données suivant une distribution non uniforme implique une répartition des activations suivant cette distribution au sein de la mémoire, et une incidence sur la phase de récupération. À travers un faible ensemble de neurones activés, le réseau est incapable de décider quel neurone devrait être activé, ce qui conduit à une ambiguïté. La section suivante rappelle comment cette question a été traitée dans la littérature.

### 4.2.1 Solutions de la littérature

La conception du réseau CbNN a entraîné des contributions portant sur l’efficacité de la gestion de cette mémoire associative [51] [78]. Cependant, les expérimentations ne concernent que la récupération de séquences binaires simulées, uniformes et de faible dimension. L’objectif du projet SENSE, apprendre des images à l’aide du CbNN, révèle l’incapacité du réseau à gérer des données non-uniformes. Désormais, les principales contributions de la littérature portent sur la problématique d’ambiguïté et sont validées avec des données artificielles suivant une distribution gaussienne, soit non-uniforme.

Les données non-uniformes influent le remplissage de la matrice d’adjacence qui suit la distribution des données. Par conséquent, seul un sous-ensemble des neurones capture la totalité des informations. Étant donné que le réseau excelle en terme de récupération sur des données uniformes, une solution consiste à utiliser un codage de Huffman [19] sur les données non-uniformes soumises à apprentissage afin de les transposer dans un espace plus homogène. La solution s’appuyant sur le codage de Huffman crée une uniformisation des données qui se répercute dans la répartition des activations du réseau. Par conséquent, l’efficacité du codage améliore les performances de la phase de récupération. Cependant, la méthode nécessite le stockage complémentaire du dictionnaire des mots de codes pour le décodage des messages retrouvés.

Une méthode de désambiguïsation consiste à ajouter une couche de codage supplémentaire en étiquetant les cliques apprises [91]. Ces étiquettes codent chacune des cliques indé-

pendamment afin d'aider le réseau à dissocier les cliques au cours de la phase de récupération. Par conséquent, si la connexion d'une nouvelle clique est commune avec une ancienne, les deux connexions porteront la nouvelle étiquette.

Une autre manière de réduire les ambiguïtés au sein du réseau s'appuie sur la connaissance de leurs apparitions. En effet, comme souligné dans le chapitre précédent, les ambiguïtés proviennent du faible nombre de neurones disponibles pour coder des informations différentes. La conséquence est la difficulté du réseau à distinguer les informations puisque toutes les données apprises sont similaires, *c.-à-d.* les mêmes neurones codent l'ensemble des informations. Par conséquent, une solution consiste à cloner les neurones les plus couramment utilisés dans le codage des informations soumises à apprentissage, afin de partager les cliques à l'aide de leurs clones [142].

Une autre contribution vise à ajouter un codage supplémentaire à travers une seconde couche de CbNN [35]. Le nouveau réseau à deux couches peut être considéré comme une machine de Boltzmann restreinte (RBM). En effet, l'activation des neurones en seconde couche est répartie uniformément en fonction des données non-uniformes de la première couche. La récupération est réalisée sur les cliques connectant les neurones des deux couches.

En parallèle, nous proposons un CbNN pourvu de plasticité possédant des caractéristiques différentes du réseau à clone [142] ou le modèle s'inspirant des machines de Boltzmann restreinte [35]. En effet, la différence provient des interactions entre les couches et la gestion des informations apprises.

## 4.2.2 *Modèle à couches multiples*

L'usage d'un modèle à couches multiples apporte une plasticité au réseau permettant de soulager les couches surchargées. En effet, la probabilité d'apparition d'une ambiguïté dépend de la densité du réseau. Autrement dit, si la densité du réseau est élevée, le taux de succès de la récupération est réduit. Nous proposons une méthode de transfert d'une couche à une autre à travers la conception d'un modèle composé de deux couches. Le modèle est par la suite généralisé à N-couches.

Le cadre d'expérimentation est restreint à des données simulées, donc générées aléatoirement suivant une distribution uniforme en premier lieu afin d'illustrer la réduction d'ambiguïtés à travers la plasticité. Puis nous utilisons des données tirées suivant une distribution gaussienne (non-uniforme), afin de valider notre modèle sur d'autres types de données et le comparer aux autres méthodes de la littérature.

### 4.2.2.1 *Un réseau à double couche*

Le but de ce nouveau réseau n'est pas seulement d'empiler deux couches, mais de les faire interagir. Le modèle est composé de deux CbNN possédant la même architecture afin

d’assurer le passage sans perte d’informations entre les couches. L’interaction entre les deux couches est réalisée à travers le transfert d’un sous-ensemble des cliques apprises dans la première couche vers la seconde.

La sélection des cliques devant être transférées repose sur les neurones contenant le plus d’information. Cette stratégie vise à réduire le phénomène d’ambiguïté. Par conséquent, il est nécessaire de séparer les neurones les plus sollicités afin de réduire des conflits qui conduiraient à une ambiguïté. À cet égard, notons  $\tau$  le paramètre de seuil basé sur la fréquence d’utilisation d’un neurone. Ce paramètre de seuil permet de sélectionner les neurones destinés à être transférés vers la couche supérieure. L’efficacité du réseau diminue lorsque sa densité augmente. Par conséquent, il est primordial d’effectuer le transfert avant d’atteindre la limite de densité  $\delta$  de la première couche, sinon la récupération peut s’avérer impossible. Rappelons que la limite de densité a été déterminée dans le chapitre précédent.

Soit  $C_i^{(L)}$  le cluster  $i$  dans la couche  $L$ ,  $v_{i,j}^{(L)}$  le  $j^{\text{ème}}$  neurone du  $i^{\text{ème}}$  cluster dans la couche  $L$ , et  $\zeta^{(L)} = (N, E)$  l’ensemble des cliques dans la couche  $L$  avec  $E$  les connexions de ces cliques à l’aide des  $N$  neurones disponibles dans le réseau. Afin de déterminer la fréquence du neurone  $v_{i,j}^{(1)}$ , nous calculons ses occurrences  $\theta_{v_{i,j}^{(1)}}$  à travers la diagonale de la matrice d’adjacence. Cette dernière est calculée non plus comme une union des cliques (équation 3.1), mais comme la somme des cliques à apprendre. Par conséquent, la matrice d’adjacence perd sa binarité et nous utilisons la règle additive suivante

$$W = \sum_i A_i. \quad (4.1)$$

Les entrées de la matrice  $W$  ne sont plus binaires mais entières avec, pour chaque neurone, son nombre précis de connexions et son occurrence sur la diagonale. Si la densité de la première couche atteint la limite  $\delta$ , les neurones dépassant le seuil  $\tau$  sont sélectionnés. Les cliques constituées de ces neurones sont transférées dans la couche suivante. Le choix du paramètre  $\tau$  définit le nombre de cliques transférées vers la seconde couche. Si  $\tau$  est nul, toutes les cliques passeront en seconde couche. Le transfert revient à translater toute la première couche sur la seconde, ne résolvant pas le problème des ambiguïtés. Si le seuil est défini comme la moyenne des occurrences des neurones du réseau tel que  $\tau_{mean} = \frac{1}{N} \sum_{i=1}^X \sum_{j=1}^N \theta_{v_{i,j}^{(1)}}$ , alors plusieurs neurones seront sélectionnés et toutes leurs cliques associées seront apprises en seconde couche. Par conséquent, la matrice d’adjacence se remplira plus rapidement au cours de l’apprentissage des messages, et deviendra le facteur limitant durant la phase de récupération. Afin de limiter le nombre de cliques à transférer, il est nécessaire de limiter le nombre de neurones à sélectionner pour un transfert. En conséquence, la sélection concerne le neurone possédant une occurrence maximale  $\tau_{max} = \max(\theta_{v_{i,j}^{(1)}})$ , conduisant ainsi à une réduction de cliques devant être transférées. Il est primordial de garder les deux couches homogènes vis-à-vis de leurs densités puisque le but du modèle est de diminuer l’apparition des ambiguïtés.

La figure 4.2 correspond à l'évolution des densité des réseaux à double couche en fonction de la méthode de transfert et du type de données apprises. Chaque couche est un CbNN plein composé de  $\chi = c = 16$  clusters comprenant  $l = 32$  neurones. La limite de densité pour un transfert est fixée à  $\delta = 0,5$  si les messages appris sont uniformes, sinon la limite est  $\delta = 0,2$ . Les tests de récupération utilisent la règle dynamique SoS et la règle d'activation WTA et sont effectués à travers 1 000 séries sur 100 messages appris, dont 25% des clusters sont effacés.

Comme l'illustre la figure 4.2a présentant les moyennes de densités, le double CbNN voit sa densité réduite par rapport au réseau initial (composé d'un unique CbNN), et ceci indépendamment de la méthode de transfert et de la distribution des messages. Bien que la moyenne de la densité du réseau à double couche utilisant le transfert avec le paramètre  $\tau_{mean}$  soit inférieure à celle avec  $\tau_{max}$ , la variance est plus élevée comme illustrée par la figure 4.2b. En effet, avec le paramètre  $\tau_{mean}$ , plusieurs neurones sont sélectionnés, entraînant un nombre élevé de cliques associées à transférer en seconde couche. Par conséquent, la densité de la dernière couche tend vers celle du réseau initial. Cet écart de densité entre les méthodes de transfert se répercute sur les performances de la phase de récupération dont les résultats sont présentés dans la figure 4.3. La phase de récupération commence par la première couche, puis opère sur la seconde si le réseau n'a pas été en mesure de récupérer une clique.

La figure 4.3 présente les taux d'erreur de correction des réseaux à simple et double couche en fonction du paramètre  $\tau$ . Les architectures des réseaux et paramètres de récupération sont les mêmes que ceux utilisés pour la figure 4.2.

Compte tenu de la différence de densité, les réseaux à double couche obtiennent de meilleurs performances que le réseau initial, et ceci indépendamment de la distribution des messages appris. La variance de densité entre les couches est plus faible avec la méthode de transfert  $\tau_{max}$  grâce à une meilleure homogénéité sur les couches, comme illustrée par la figure 4.2b. Le défaut de la méthode  $\tau_{mean}$  provient du transfert d'un nombre élevé de cliques sur la couche supplémentaire, ce qui entraîne un pic d'ambiguïtés visible sur la figure 4.3a après l'apprentissage de 1 500 messages. Cependant, le transfert d'une couche à une autre à l'aide de la méthode  $\tau_{max}$  permet de séparer plus délicatement les cliques, résultant alors en une amélioration des performances. La figure 4.1 illustre ces phénomènes.

Dans le cas d'expérimentations sur les données non-uniformes, le seuil de densité utilisé pour limiter l'erreur de récupération est réduit de  $\delta = 0,5$  à  $\delta = 0,2$ , soit la limite empiriquement définie dans le chapitre précédent. En dépit d'une densité plus faible du fait de l'apprentissage de données gaussiennes, le nombre de messages stockés au sein de la mémoire avant que le réseau soit incapable de récupérer une information est divisé par deux. L'apprentissage par le réseau à clique de données non-uniformes entraîne une apparition accrue des ambiguïtés (illustrée par la pente des courbes visible dans les figures 4.3a

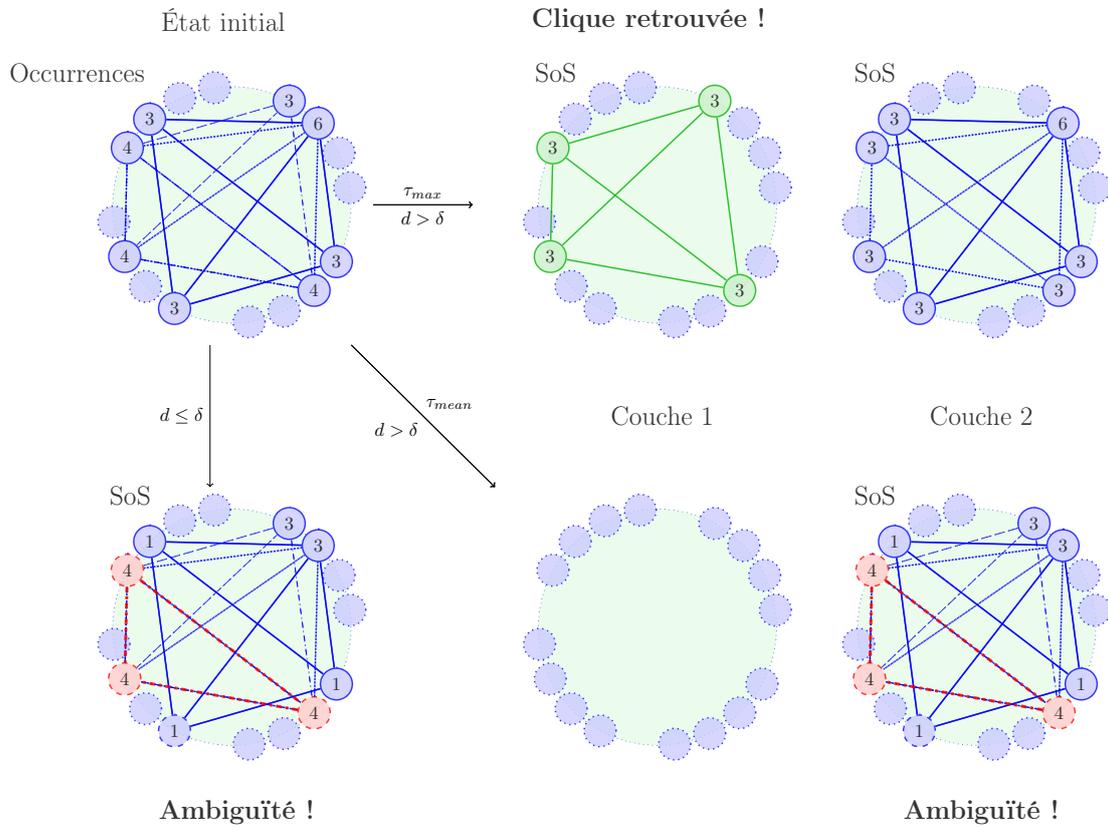


FIGURE 4.1 – Exemples d’utilisation du modèle à multiples couches de CbNN.

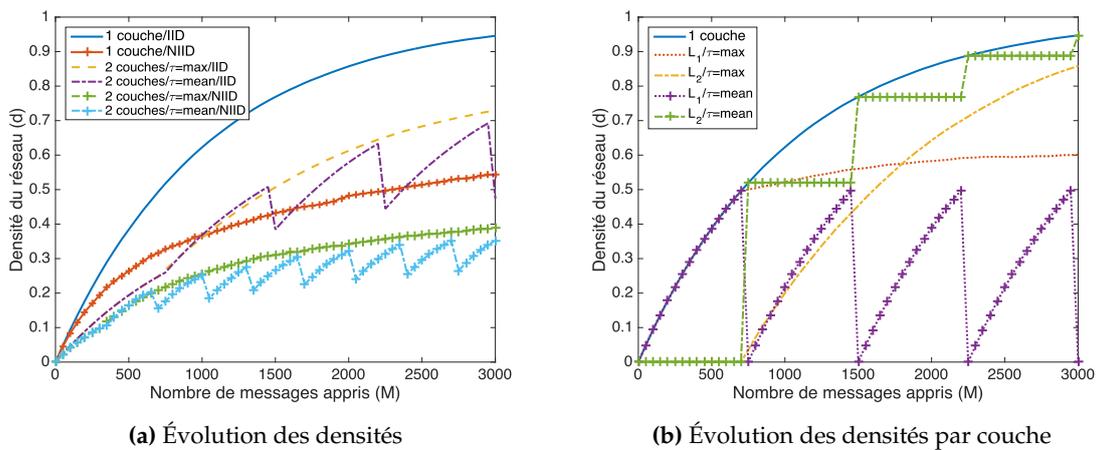
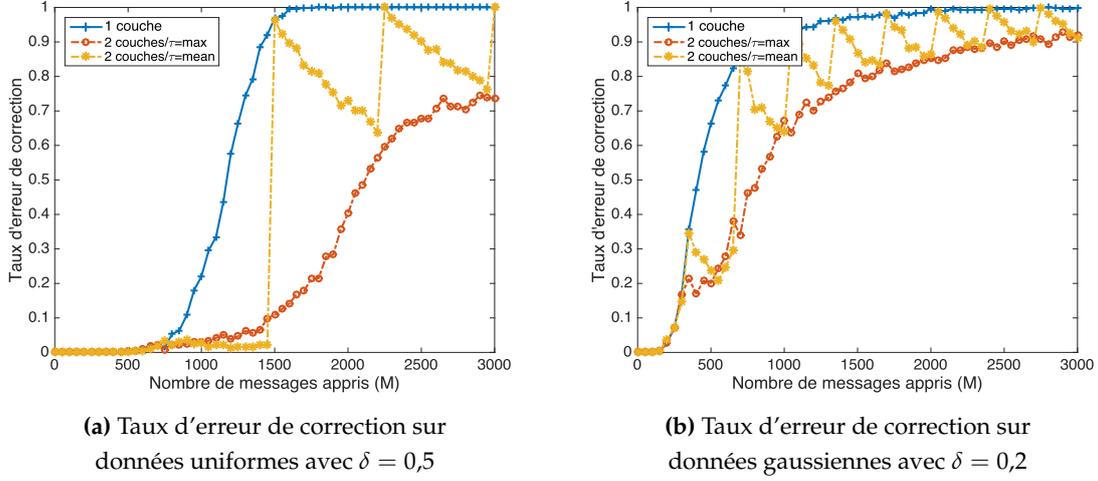


FIGURE 4.2 – Évolution des densités des réseaux à deux couches en fonction du paramètre  $\tau$  rapport à un modèle simple couche.

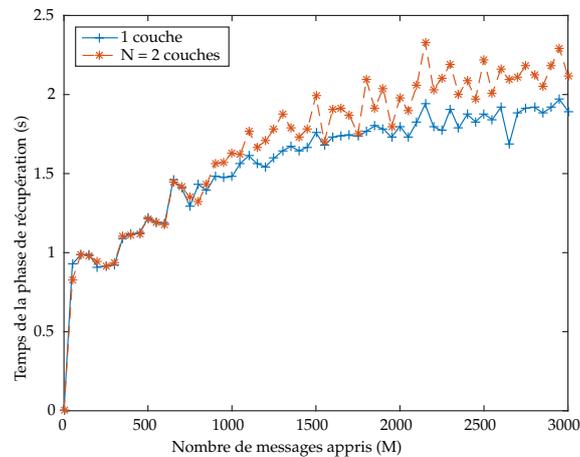


**FIGURE 4.3** – Taux d'erreur de correction des réseaux à deux couches en fonction du paramètre  $\tau$  et de la distribution des données apprises par rapport à un modèle simple couche.

et 4.3b). L'inclinaison plus importante de la courbe pour les données non-uniformes provient de l'utilisation répétée des mêmes neurones pour représenter l'information. Bien que l'utilisation de couches supplémentaires permette un gain de performances, le réseau nécessite de la mémoire supplémentaire après un certain nombre de messages appris. En effet, la dissociation des cliques dans la phase d'apprentissage nécessite de conserver les cliques dans une matrice complémentaire à la matrice d'adjacence. Par conséquent, deux méthodes peuvent être employées avec un tel modèle. La première méthode (utilisée dans les expériences) consiste à stocker les cliques dans une matrice complémentaire à la matrice d'adjacence. La seconde méthode s'appuie sur le remplissage de la matrice d'adjacence suivant la règle additive (équation 4.1). La dissociation des cliques durant la phase d'apprentissage est entreprise à travers la factorisation de la matrice d'adjacence. En effet, à l'instar de l'équation (4.1), le remplissage de la matrice d'adjacence  $A$  peut s'effectuer en dérivant l'équation  $A_i = T_i^t T_i$  en  $A = T^t T$  tel que

$$W_{i,j} = \sum_{k=1}^m T_{i,k} T_{k,j}. \quad (4.2)$$

Une étude de la factorisation de matrice afin de récupérer l'ensemble des cliques à partir de la matrice d'adjacence est proposée dans l'annexe A. Cette méthode requiert une consommation mémoire supérieure, mais qui reste moindre comparé à la méthode de remplissage des couches du réseau CbNN initial. En effet, le réseau à une seule couche apprend tous les messages, impliquant un remplissage continu de sa matrice d'adjacence, et donc une augmentation des "1" liés aux connexions. Cependant, à l'instar de la figure 4.2a, le remplissage du réseau à deux couches diffère par la réduction de la redondance d'information, et donc la diminution de la consommation mémoire par rapport au modèle initial.



**FIGURE 4.4** – Évolution du temps de calcul pour la phase de récupération d’un réseau à simple couche et à deux couches ( $\tau_{max}$ ).

Ces expérimentations se sont focalisées sur les performances de récupération d’un réseau à deux couches afin de justifier l’importance de la plasticité dans la réduction des ambiguïtés. L’avantage du réseau à deux couches est l’efficacité de la phase de récupération qui est identique ou meilleure que le modèle à simple couche, du fait de la faible densité des couches. Toutefois, plus le réseau apprend de messages, et plus la densité augmente sur les deux couches. La phase de récupération est étendue à la seconde couche si le message recherché n’est pas retrouvé dans la première. Par conséquent, le temps de récupération peut être plus élevé pour le réseau à deux couches comme l’illustre la figure 4.4. Cependant, la différence de temps est relative, car en moyenne le réseau à deux couches nécessite 1,72 secondes contre 1,58 pour le simple couche afin de retrouver l’ensemble des messages de test. La différence du temps de calcul est de 0,14 secondes et ce pour une moyenne d’erreur de récupération de 27,1% contre 62,4% pour le modèle initial simple couche. Autrement dit, le réseau à double couche obtient un gain moyen supérieur à 50% d’efficacité de correction pour seulement 8,9% de temps de récupération moyen en plus. La plasticité permet au nouveau modèle d’être plus performant que le modèle initial. Par conséquent, la réduction d’ambiguïtés dépend de la densité du réseau. Les résultats de la figure 4.4 s’appuient sur la même architecture que les résultats des figures précédentes relatives au modèle double couche.

Bien que l’usage d’une seconde couche permette d’apprendre davantage de messages avant que le réseau ne soit dans l’incapacité de récupérer une information, la densité de la seconde couche sera au final plus importante que celle de la première couche. Afin de limiter le stockage sur la seconde couche, la plasticité peut être étendue à N couches avec le seuillage des densités associées permettant une récupération optimale.

#### 4.2.2.2 Extension à $N$ couches

Les performances d'un modèle à plusieurs couches dépendent de l'apprentissage. En effet, si ce dernier sature la mémoire de messages, elle n'aura pas le temps d'effectuer les sauts de couches. Cependant, si ces sauts sont effectués régulièrement, les performances seront améliorées. Par conséquent, les performances obtenues étant donné un nombre de messages dépendent du nombre de couches présentes dans le modèle.

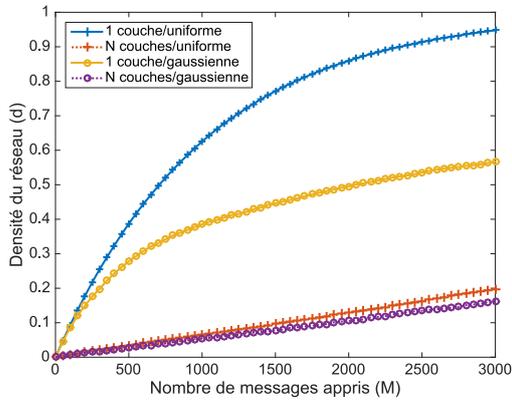
Le réseau est étendu avec l'ajout d'une nouvelle couche lorsque la densité de la couche initiale dépasse son seuil limite  $\delta$ . Les mécaniques du réseau à  $N$  couches restent identiques à celles du modèle à double couche. Autrement dit, dès lors qu'une couche est surchargée, une partie de ses cliques sont transférées vers une couche supérieure. L'extension du modèle à  $N$  couches permet de stabiliser le taux de correction en dépit de la mémoire et du temps nécessaire pour la récupération, notamment en présence de données non-uniformes. De même que pour le réseau précédent, nous étudions les capacités de stockage et de récupération du modèle à  $N$  couches avec des données artificielles générées suivant une distribution uniforme et gaussienne. Comme lors des expérimentations précédentes, la densité est réduite si les données apprises sont non-uniformes. En outre, la plasticité permet une évolution linéaire de la densité et par conséquent réduit l'erreur de récupération par rapport au réseau composé d'une unique couche.

Le désavantage du réseau à  $N$  couches concerne le temps de récupération de l'information au sein de l'ensemble des couches si la phase de récupération est réalisée séquentiellement sur les couches. Cependant, puisque ces dernières sont indépendantes les unes des autres, la phase de récupération peut être effectuée en parallèle à plus large échelle.

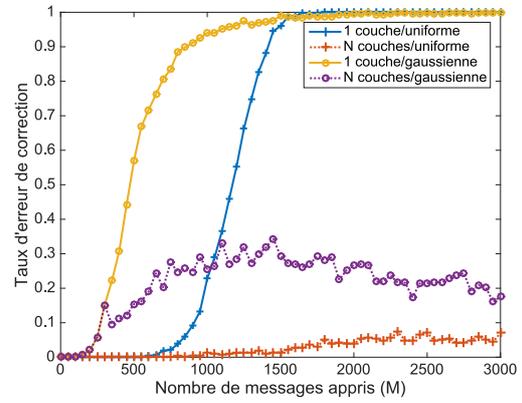
Les réseaux employés dans les expérimentations possèdent la même architecture pleine, soit  $\chi = c = 16$  clusters comprenant chacun  $l = 32$  neurones. La limite de densité pour un transfert est la même pour les deux distributions et est définie à  $\delta = 0,2$ . Le paramètre de transfert est  $\tau_{max}$ . Les expérimentations s'appuient sur des tests de récupération de messages partiellement effacés (25% d'effacement). Dans le cas d'autres types de dégradation, le réseau est confronté au problème de généralisation comme nous le verrons plus loin dans ce mémoire.

La figure 4.5a donne la moyenne des densités sur l'ensemble des couches pour chaque réseau. Étant donné que la limite de densité est identique quelle que soit la distribution des données, cette figure permet d'illustrer la répartition homogène de l'information. En effet, nous observons une tendance identique pour l'évolution des densités associées aux différents types de données.

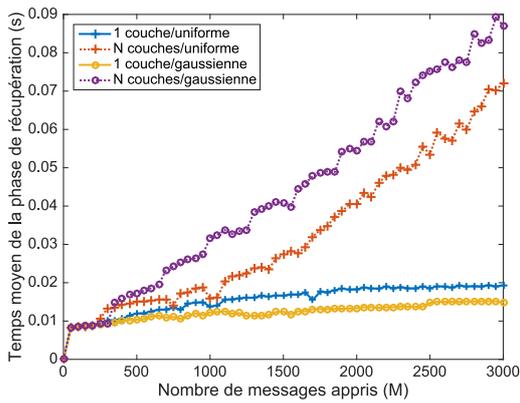
En ce qui concerne les erreurs de récupération, la figure 4.5b illustre les excellentes performances du modèle à  $N$  couches, et ce que les messages appris soient uniformes ou non. Les tests de récupération correspondent à une moyenne de 1 000 séries sur des messages ap-



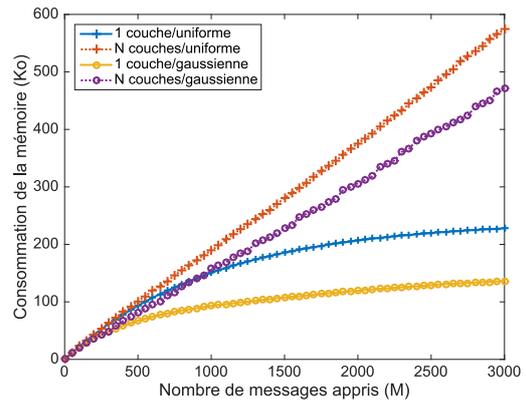
(a) Evolution des densités



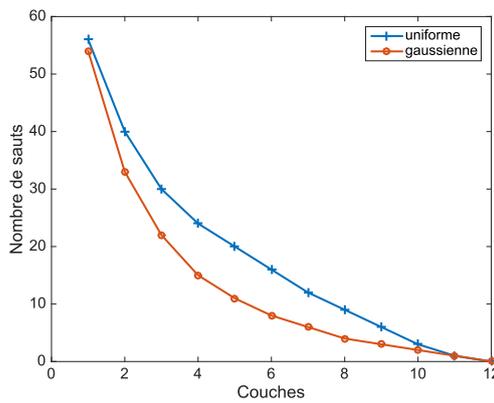
(b) Taux d’erreur de correction



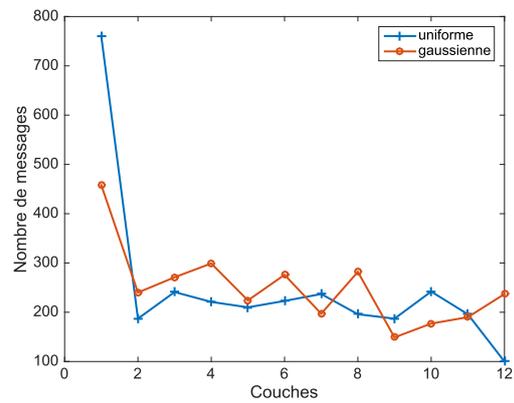
(c) Temps moyen d’une récupération



(d) Consommation de la mémoire



(e) Nombre de transferts effectués par couche



(f) Distribution des messages par couche

FIGURE 4.5 – Résultats d’expérimentations sur des données suivant une distribution uniforme et gaussienne avec des réseaux à simple couche et à couches multiples.

pris en utilisant la règle dynamique SoS et la règle d'activation WTA. Nous considérons que la phase de récupération est un succès si et seulement si la totalité du message a été retrouvée. La métrique utilisée pour les tests de récupération s'appuie sur la distance de Hamming calculée entre le message appris sans erreur et celui récupéré par le réseau. Par conséquent, si le message est retrouvé, l'erreur de récupération est nulle, sinon elle est égale à 1. Les courbes de la figure 4.5b correspondent aux moyennes des erreurs de chaque modèle.

Pour les données uniformes, le modèle à couches multiples ne dépasse pas 10% d'erreur lors de nos expérimentations, alors que le modèle à simple couche n'est plus en mesure de corriger un message dégradé après 1 500 messages appris. Tandis qu'après 3 000 messages appris, le réseau à double couche atteint 70% d'erreur, le modèle à N couches ne dépasse pas 10% d'erreur. Par conséquent, l'erreur de récupération des messages non-uniformes est accrue avec un faible nombre de couches, mais elle décroît grâce à la plasticité du réseau qui s'étend continuellement pour homogénéiser l'information sur l'ensemble des couches. Cependant ces performances ont un coût. En effet, plus il y a de couches, et plus le temps de récupération d'un message est élevé, comme illustré par la figure 4.5c. Avec des données uniformes, le temps est multiplié par 3,73 en passant du modèle simple couche à un modèle à N = 12 couches. Pour les données non-uniformes, le facteur est de 5,84. Cette différence prend en compte le nombre de couches supérieures, et la récupération d'un message non-uniformes qui requiert davantage de temps. En effet, la récupération est réalisée séquentiellement en démarrant par la dernière couche jusqu'à trouver le message recherché.

La plasticité a pour effet d'augmenter le temps de récupération, mais aussi le coût mémoire du réseau. La figure 4.5d correspond au nombre d'octets d'information relatifs aux matrices d'adjacence des divers réseaux. La consommation de la mémoire est la plus faible pour le modèle à simple couche avec des données non-uniformes. En effet, les neurones activés pour représenter les informations apprises concernent seulement un sous-ensemble de la totalité des neurones du réseau. Cependant, l'activation des neurones par couche, bien que leur nombre soit réduit par l'expansion du réseau, nécessite le stockage des matrices d'adjacence associées. Toutefois, l'impact de la consommation mémoire induite par les matrices d'adjacence est négligeable du fait de la faible dimension de l'architecture (nombre de clusters et de neurones) des réseaux. En effet, ces dimensions sont inhérentes à celles des données à apprendre. Les différences de densité et de consommation mémoire peuvent aussi être expliquées par le nombre de transferts effectués entre les couches, tel que l'illustre la figure 4.5e. Le nombre de sauts entre deux couches avec des données uniformes est plus élevé à cause du remplissage des matrices d'adjacence qui dépend de la distribution des données. En effet, les données uniformes remplissent plus rapidement la matrice d'adjacence que les données non-uniformes.

La figure 4.5f illustre l'effet de poussée au sein du réseau. En effet, tous les messages passent par la première couche avant de pouvoir être transférés vers les couches supérieures.

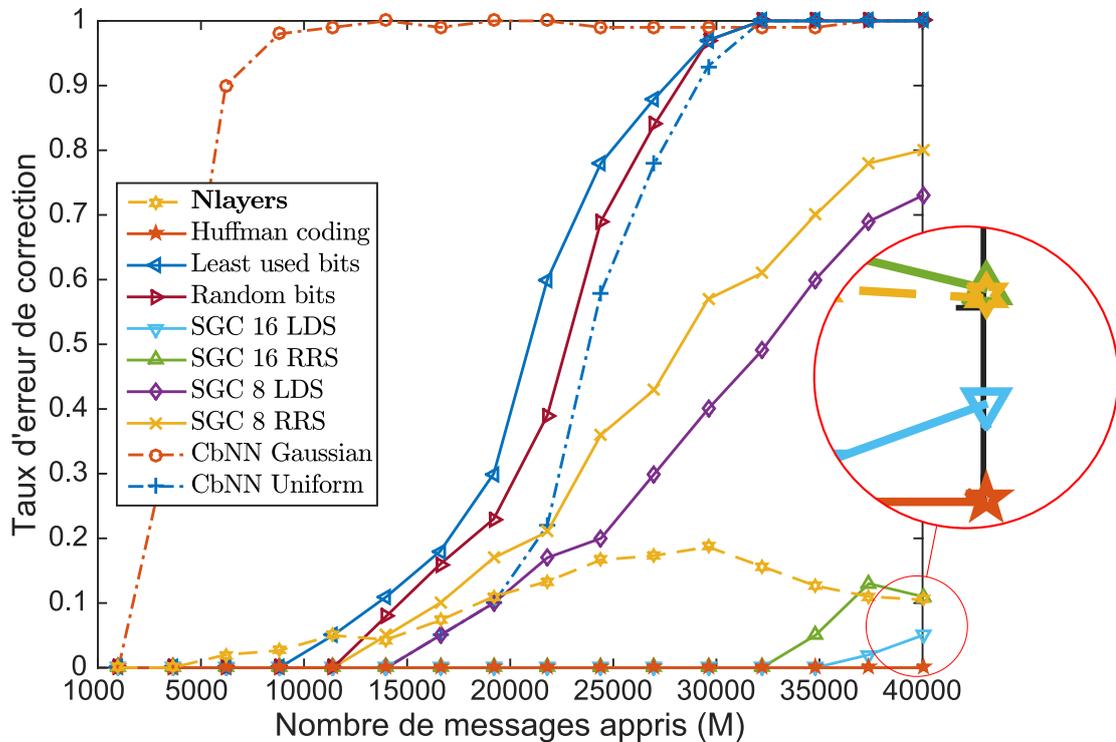


FIGURE 4.6 – Taux de récupération d’un réseau à couches multiples comparé aux méthodes de la littérature [142].

Les résultats de la figure 4.5 s’appuient sur la même architecture du réseau à double couche à la différence près que le nombre de couches par réseau n’est plus limité à deux.

La figure 4.6 positionne le modèle à couches multiples par rapport aux autres méthodes de la littérature. Le protocole d’expérimentation s’appuie sur l’apprentissage de données non-uniformes. La dimension de ces données est de 64 bits. Ainsi, les réseaux comportent  $\chi = 8 = c$  clusters et  $l = 256$  neurones chacun. Les méthodes sont comparées aux modèles initiaux de CbNN suivant une distribution uniforme et gaussienne  $\mathcal{N}(125,25)$ . Les résultats de correction sont les moyennes sur 1 000 récupérations de messages partiellement effacés à hauteur de 50% des clusters actifs.

Toutes les méthodes reposent sur une certaine forme de plasticité. En effet, notre contribution et celle des clones (ou *symmetric CbNN with global cloning, SGC*) [142] s’appuient sur des réseaux composés de plusieurs couches. Notre contribution étend le réseau en fonction des messages appris. La méthode des clones nécessite un réseau possédant plusieurs sous-réseaux contenant les clones qui permettent la désambiguïté. En l’occurrence, parmi les quatre réseaux à clones, une architecture repose sur l’utilisation de 8 sous-réseaux, et l’autre (la plus performante) sur 16 sous-réseaux. Outre la différence liée au nombre de

sous-réseaux, les réseaux à clones se distinguent par leurs méthodes d'activation. En effet, la plasticité dans ces réseaux se traduit par l'activation d'un clone dans un sous-réseau afin de dissocier les cliques apprises pour un même neurone. Le principe de *Round Robin Selection* (RRS) [142] consiste à sélectionner le sous-réseau le moins utilisé récemment. Le principe de *Least Dense Selection* (LDS) [142] cherche à simuler une compétition entre les neurones, et consiste à sélectionner le meilleur sous-réseau à travers un score qui lui est associé. Le sous-réseau possédant le meilleur score est sélectionné, et le clone du neurone associé est activé.

Les méthodes *Least used bits* [19] et *Random bits* [19] procèdent par ajout d'informations à chaque cluster et non uniquement à la clique. Afin d'obtenir des performances similaires aux données uniformes, deux bits sont ajoutés à chaque cluster, augmentant ainsi la dimension des données de 64 bits à 80 bits. Les architectures associées sont des CbNN comportant  $\chi = 8 = c$  clusters et  $l = 1024$  neurones chacun. Le coût mémoire est équivalent à 16 réseaux CbNN traitant les données de 64 bits, et donc des architectures équivalentes aux méthodes à clones. En ce qui concerne la méthode de codage de Huffman, elle requiert aussi une addition de bits, portant la dimension à 80 bits, et elle nécessite un réseau de dimension égale aux réseaux des méthodes *Least used bits* et *Random bits*. Un désavantage du codage de Huffman est le stockage d'une matrice supplémentaire (le dictionnaire) afin de coder et décoder les cliques. De plus, à cause de ce dictionnaire, les cliques ne peuvent pas être apprises de façon continue dans le réseau. En effet, l'apprentissage nécessite de connaître au préalable toutes les cliques qui doivent être apprises.

La méthode reposant sur le codage de Huffman reste la plus performante en terme de correction grâce à sa maximisation de la distance entre les bits des mots de codes et donc l'activation des neurones. Cependant, elle nécessite de connaître l'ensemble des cliques au préalable, et un espace de stockage supplémentaire pour le dictionnaire. Par conséquent, la méthode la plus efficace est la méthode à clones, bien qu'elle nécessite l'utilisation de 16 sous-réseaux. Notre contribution reste compétitive en termes de correction, mais nécessite l'utilisation de 66 réseaux en parallèle. En effet, le seuil de densité est limité à  $\delta = 0,05$  pour déclencher le transfert de cliques d'une couche à une autre.

#### 4.2.2.3 Conclusion

Les ambiguïtés apparaissent quand le réseau est incapable de dissocier les connexions des neurones, empêchant ainsi la récupération d'une clique. En effet, la connexion entre deux neurones peut être partagée entre plusieurs messages. L'expansion du réseau en plusieurs couches permet de dissocier les messages, afin de limiter le partage de connexions des neurones. Le modèle à couches multiples améliore les performances, que ce soit avec des données uniformes ou non. La force de ce type de réseau réside dans sa généralisation à N couches, permettant de conserver d'excellentes performances au détriment d'un accrois-

Méthode	Dimension des données	Architectures
<b>Nlayers</b>	<b>64 bits</b>	<b>(256,8,66)</b>
Huffman coding	80 bits	(1024,8,1) ~ (256,8,16)
Least used bits	80 bits	(1024,8,1) ~ (256,8,16)
Random bits	80 bits	(1024,8,1) ~ (256,8,16)
SGC 16 LDS	64 bits	(256,8,16)
SGC 16 RRS	64 bits	(256,8,16)
SGC 8 LDS	64 bits	(256,8,8)
SGC 8 RRS	64 bits	(256,8,8)
CbNN Gaussian	64 bits	(256,8,1)
CbNN Uniform	64 bits	(256,8,1)

**TABLEAU 4.1** – *Évaluation des méthodes de la littérature.*

sement du temps de récupération et de la consommation mémoire.

L’inconvénient d’un tel réseau est la nécessité de connaître les cliques apprises que ce soit à l’aide d’un stockage complémentaire ou par la factorisation des matrices d’adjacence. L’obtention des cliques à travers ces méthodes rend inutile l’inférence du CbNN pour récupérer une clique. En effet, au lieu d’utiliser les mécaniques du CbNN, nous pouvons utiliser une méthode de plus proches voisins avec une distance de Hamming afin de garantir la récupération optimale d’une clique. Cependant, le modèle à couches multiples démontre l’importance de conserver une densité minimale dans le réseau CbNN afin de maximiser son taux de récupération sans erreur, et ce d’autant plus si les messages suivent une distribution non-uniformes.

Le modèle à couches multiples résout en partie le problème de l’ambiguïté en soulageant la charge des neurones à travers la dissociation des cliques. La dissociation est réalisée à travers le transfert des cliques vers d’autres couches afin d’homogénéiser la densité du réseau. Autrement dit, la gestion de l’ambiguïté est assurée par une modification de l’architecture du CbNN. Une alternative à cette modification consiste à changer le calcul des règles utilisées lors de la phase de récupération. Dans la partie suivante, nous présentons cette alternative sous forme d’un terme de pénalisation.

### 4.2.3 Pénalisation

Le processus de récupération du réseau, détaillé dans le chapitre précédent, est itératif et s’appuie sur un critère d’arrêt tel qu’un nombre maximal d’itérations ou la production d’une clique [2]. Ce dernier critère décide qu’une clique est retrouvée si et seulement si les scores des neurones la représentant sont égaux. En effet, une clique étant un graphe com-

plet, tous les neurones la composant sont donc connectés, impliquant l'égalité dans leurs scores calculés par une règle dynamique. Cependant, en présence d'ambiguïtés, cette dernière configuration devient improbable. Afin de surmonter ce problème, nous proposons un nouveau terme de pénalisation s'appuyant sur la définition d'une clique. Nous visons à promouvoir les neurones ayant les mêmes scores que ceux qui leur sont connectés afin de suggérer une clique correspondante aux données soumises. Pour un neurone  $i$ , nous définissons sa similarité aux autres neurones comme la somme des différences entre son score  $\lambda_i$  et les scores des autres neurones. Le score est issu d'une règle dynamique telle que le SoS. La similarité est calculée comme

$$s_i = \sum_{j=1, j \neq i}^n W_{j,i} |\lambda_j - \lambda_i|. \quad (4.3)$$

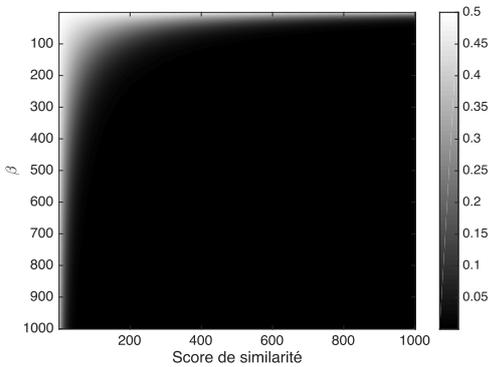
Ce score peut être normalisé en sommant tous les scores de chaque neurone d'un même cluster, soit  $\hat{s}_i = s_i / \sum_{k \in c_i} s_k$  avec  $c_i$  désignant tous les neurones du cluster auquel le neurone  $i$  appartient. Ce score de similarité entre deux neurones sera nul si ceux-ci sont issus du même cluster (cf. à cause de la contrainte structurelle). Si la similarité tend vers 0 ( $s_i \rightarrow 0$ ), alors la probabilité qu'ont les neurones connectés d'appartenir à une seule clique est élevée. A contrario, si la similarité tend vers l'infini ( $s_i \rightarrow \infty$ ), le neurone partagera des connections avec d'autres neurones de cliques différentes. Finalement, afin de favoriser la dissociation, et donc privilégier les neurones qui ont un score de similarité proche de 0, nous utilisons la fonction sigmoïde sur ces scores

$$\tilde{\lambda}_i = \frac{\lambda_i}{1 + e^{\beta s_i}}, \quad (4.4)$$

où  $\beta > 0$  est un paramètre qui gouverne la force de l'a priori sur le comportement de la convergence du réseau. La positivité de  $\beta$  et celle du score de similarité de l'équation (4.3) permet de maximiser la pénalisation (équation 4.4), pour des faibles valeurs de ces paramètres.

La figure 4.7a illustre les effets du paramètre  $\beta$  et de la fonction sigmoïde dans la pénalisation (équation 4.4) sur des scores de similarité  $s_i$  d'un neurone  $i$  (équation 4.3). Le tableau 4.7b de la figure 4.7 indique les tendances de la pénalisation en fonction des scores de similarité et la force de l'a priori. Ainsi, l'utilisation de cette fonction (telle que  $s_i \rightarrow 0$ ) pousse les neurones à obtenir des scores finaux  $\tilde{\lambda}_i$  qui tendent vers la moitié de leurs scores initiaux  $\lambda$  (tableau 4.7b). Quant aux neurones caractérisés par  $s_i \rightarrow \infty$ , la fonction sigmoïde fait tendre leurs scores  $\tilde{\lambda}_i$  vers 0 (tableau 4.7b). Par conséquent, ce nouveau terme de pénalisation tend à promouvoir à chaque itération des solutions où les neurones qui ont un score similaire sont connectés en s'appuyant sur la matrice d'adjacence. Autrement dit, la pénalisation permet d'atténuer le problème d'ambiguïté jusqu'à une densité plus large de l'information dans le réseau, où celui-ci n'est plus en mesure de corriger l'information.

Nous montrons avec l'illustration 4.8 les scores mis à jour par la pénalisation. La figure 4.8a est reprise du chapitre précédent, présentant la problématique d'ambiguïté. La figure

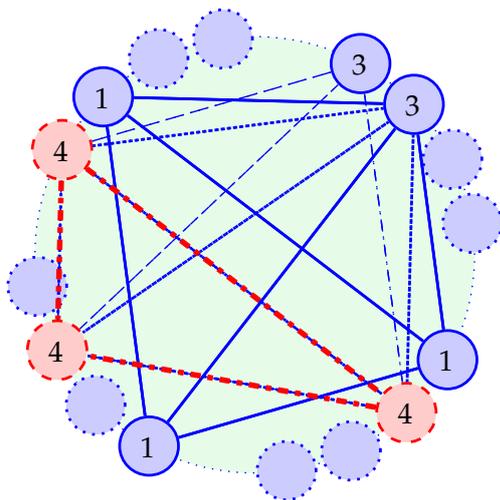


(a) Effet de la pénalisation

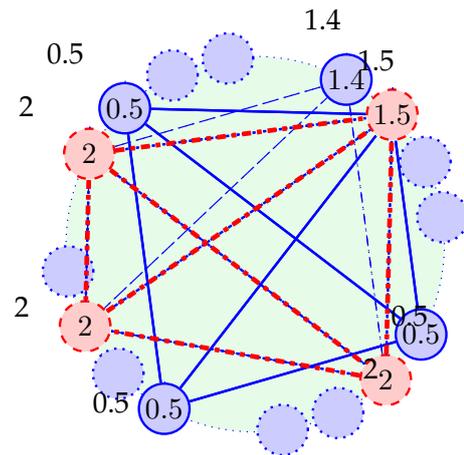
	$s_i \rightarrow 0$	$s_i \rightarrow \infty$
$\beta \rightarrow 0$	$\tilde{\lambda}_i \rightarrow \frac{\lambda_i}{2}$	$\tilde{\lambda}_i \rightarrow 0$
$\beta \rightarrow \infty$	$\tilde{\lambda}_i \rightarrow 0$	$\tilde{\lambda}_i \rightarrow 0$

(b) Limites

FIGURE 4.7 – Les effets du paramètre  $\beta$  dans la pénalisation.



(a) Scores SoS issus de la correction du message  $\{_{-}4,2,3\}$



(b) Scores SoS avec pénalisation issus de la correction du message  $\{_{-}4,2,3\}$

FIGURE 4.8 – Application de la pénalisation sur un exemple d’ambiguïté.

4.8b reprend le même exemple, mais avec les scores calculés à l’aide de la pénalisation. Le réseau est composé de  $\chi = 4$  clusters comprenant chacun  $l = 4$  neurones. Les messages appris sont  $\{3,3,4,4\}$  (solide),  $\{4,4,2,3\}$  (tirets) et  $\{3,4,2,3\}$  (pointillés). La règle dynamique de la phase de correction est le *Sum-of-Sum* (SoS) sans et avec pénalisation (gauche et droite respectivement). Nous pouvons observer que la pénalisation aide le réseau à sélectionner le neurone à activer afin d’éviter une ambiguïté.

### 4.2.3.1 Expérimentations sur des données artificielles

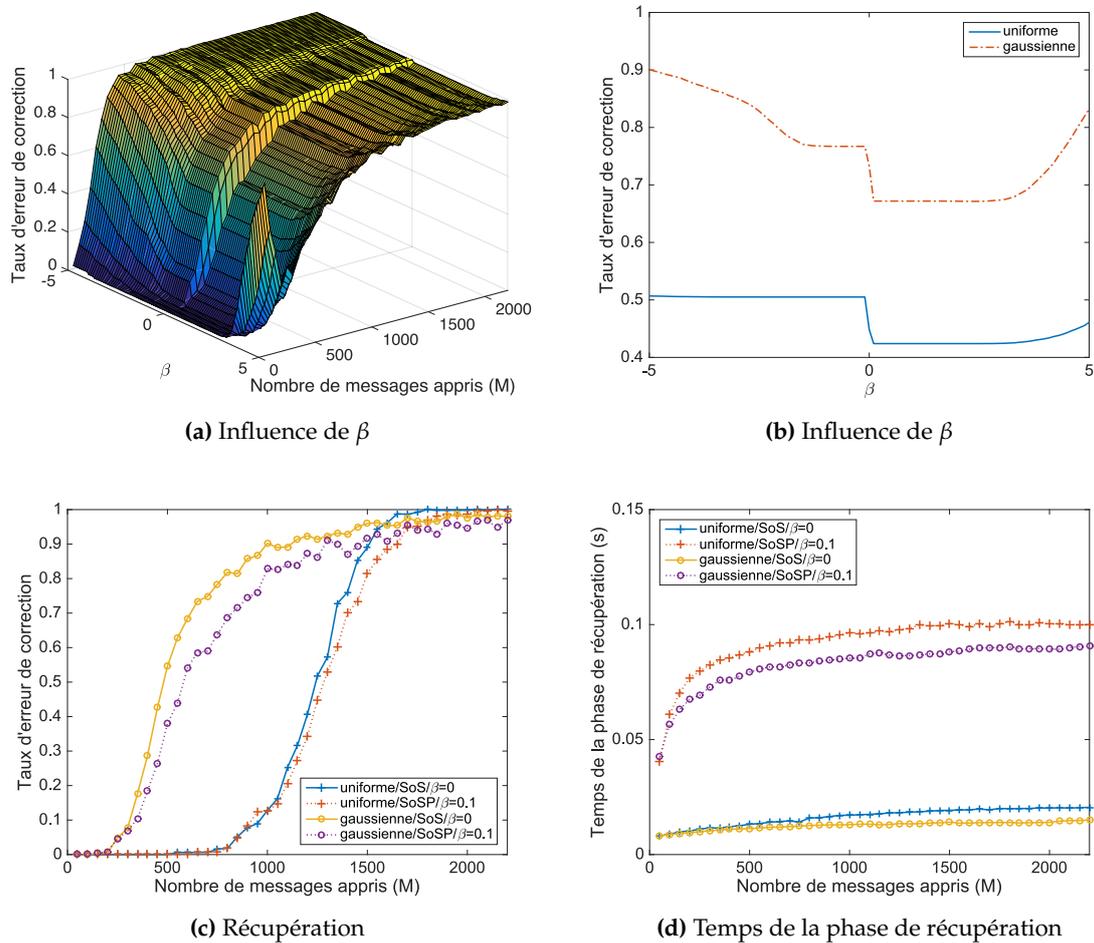
L'évaluation de la stratégie proposée pour la désambiguïsation est conduite avec un dispositif expérimental où les données apprises sont partiellement effacées et doivent être corrigées par le CbNN. En effet, sachant le réseau dépourvu de capacité de généralisation, nous souhaitons corriger uniquement des données apprises au préalable afin d'évaluer si la pénalisation peut gérer l'ambiguïté. La première partie des expérimentations s'appuie sur des données générées aléatoirement suivant une distribution uniforme qui peuvent présenter des ambiguïtés à moindre échelle, puis nous expérimentons avec des données non-uniformes. La seconde partie est consacrée aux images afin de valider le principe de pénalisation dans le contexte d'une tâche de reconstruction, montrant ainsi les prémices d'utilisation d'un CbNN en vision par ordinateur.

La figure 4.9 illustre les résultats de l'utilisation de la pénalisation sur des données uniformes et non-uniformes avec un réseau CbNN plein. L'apprentissage du réseau est réalisé à travers la soumission de messages générés aléatoirement. Puis la phase de récupération est réalisée à deux reprises, l'une avec la règle dynamique SoS classique, l'autre avec pénalisation (SoSP). L'architecture de CbNN utilisée pour les expérimentations est composée de  $\chi = c = 16$  clusters comprenant  $l = 32$  neurones chacun. Les résultats sont les moyennes de 500 séries sur des messages appris dégradés en effaçant 25% des clusters. Les courbes présentées illustrent les résultats pour les  $\beta \in [-5; 5]$  obtenant les meilleures performances.

Quel que soit le nombre de messages appris, la pénalisation permet l'amélioration des résultats. L'utilisation d'un paramètre  $\beta$  proche de 0 permet d'obtenir les meilleures performances indépendamment de la distribution des données. La surface présentée dans la figure 4.9a permet de déterminer le meilleur paramètre de pénalisation  $\beta$  par validation croisée sur des données non-uniformes. En effet, l'impact de la pénalisation sur des données uniformes est moindre du fait d'une plus faible probabilité de rencontrer une ambiguïté (figure 4.9c). La figure 4.9b montre l'influence du paramètre  $\beta$  sur la moyenne de l'ensemble des résultats : le taux d'erreur de correction est plus bas pour un  $\beta$  proche de 0.

Les résultats de récupération donnés dans la figure 4.9c démontrent que l'usage de la pénalisation permet d'apprendre davantage de messages avant d'atteindre un taux d'erreur de correction identique. Les performances sur les données non-uniformes sont plus élevées que sur les données uniformes du fait de la différence de probabilité d'apparition des ambiguïtés. La pénalisation est un processus supplémentaire appliqué sur les scores calculés par la règle dynamique SoS. Par conséquent, l'usage de la pénalisation demande un temps de calcul supplémentaire, mais celui-ci reste négligeable comme illustré par la figure 4.9d. Le temps de récupération est logarithmique en fonction du nombre de messages appris par le réseau puisque peu de neurones inactifs sont sollicités lors de l'apprentissage de nouveaux messages

Dans le cas de données artificielles, les performances de la pénalisation sont en deçà



**FIGURE 4.9** – Effet de la pénalisation : influence de  $\beta \in [-5; 5]$  sur le taux d’erreurs de correction (b) et en fonction des messages non-uniforme (a), performances de récupérations pour les meilleurs  $\beta$  (c) et temps de calcul moyen (d).

de celles obtenues par les méthodes de la littérature. En effet, quand les autres méthodes reposent sur la désambiguïté de données non-uniformes à faible dimension, la pénalisation a été pensée pour des données à forte dimension, telles que des images. Autrement dit, puisque les données traitées sont de faible dimension, l’utilisation de la plasticité est possible. Que ce soit l’ajout de données supplémentaires, nécessitant une plus large architecture, ou l’ajout de réseaux supplémentaires, le coût de ces ajouts reste faible du fait de la faible dimension des données artificielles. Cependant, dans le cas d’images, et donc de données de dimension supérieure, l’utilisation de plasticité peut être un frein comme nous allons le détailler par la suite.

### 4.2.3.2 Expérimentations sur des images

Afin d'illustrer l'utilisation du CbNN sur des images, nous considérons ici le jeu de données standard MNIST [94] présenté aussi dans l'annexe B. Celui-ci est caractérisé par une faible dimension spatiale, ce qui permet d'utiliser le CbNN directement dans l'espace des pixels et non pas après une étape d'extraction de caractéristiques. Le jeu de données est composé d'images en 8 bits de résolution  $28 \times 28$  pixels représentant les 10 chiffres manuscrits (de 0 à 9), et il est divisé en ensembles d'apprentissage et de test. Nous utilisons ici uniquement l'ensemble d'apprentissage puisque le CbNN est incapable de généraliser des informations inconnues.

L'architecture du réseau dépend du type de données traitées. En l'occurrence, nous traitons directement les pixels des images. Par conséquent, comme présenté dans le chapitre précédent, le nombre de clusters  $\chi$  est égal au nombre de pixels dans les images. L'inconvénient de cette approche est que l'architecture s'adapte aux premières données apprises, ici des images de résolution  $28 \times 28$ . Le réseau sera par la suite incapable d'apprendre des images de résolution différente (par exemple  $32 \times 32$ ). Pour ce faire, il faudra alors utiliser un autre réseau. Le nombre de neurones  $l$  dépend de la quantification appliquée à l'image afin de réduire sa dimension, comme détaillé dans le chapitre précédent. Cette quantification est réalisée par l'algorithme des K-moyennes, avec  $K$  le nombre de neurones souhaité pour chaque cluster du réseau. Les moyennes mobiles, résultat de cet algorithme, restent les mêmes dans le cas d'un apprentissage incrémental (*online*), ce qui peut ajouter un biais si le nombre de données en début d'apprentissage est restreint. Afin de minimiser ce biais, nous appliquons la méthode des K-moyennes sur l'ensemble des images que nous projetons d'apprendre. Notons que d'autres méthodes de quantifications sont possibles et que notre approche n'est pas contrainte par la stratégie de quantification choisie. Nous garderons cette même architecture pour toutes les expériences menées sur des images. Nous utilisons ici la moyenne des RMSE calculées sur les images du jeu de données.

L'évaluation des résultats est réalisée avec l'erreur quadratique moyenne (plus précisément sa racine carrée, ou RMSE) mesurée sur les images partiellement effacées qui ont été soumises au CbNN à des fins de correction. En effet, si le RMSE tend vers 0, cela se traduit par une reconstruction tendant vers l'image d'origine. Sinon, cela signifie que le réseau n'a pas été en mesure de corriger correctement l'image.

Nous comparons la règle dynamique Sum-of-Sum (SoS) standard avec la règle proposée, soit la règle SoS suivie d'une pénalisation (SoSP). La règle d'activation est le WTA si le réseau est plein et GkWTA avec  $k = c$  pour un réseau parcimonieux. Le niveau de parcimonie (le nombre de clusters actifs) a été fixé à  $c = \chi/4 = 196$  dans les expérimentations. Les résultats sont reportés dans le tableau 4.2. Dans le cas de SoSP, le paramètre  $\beta$  a été déterminé empiriquement par validation croisée. Le nombre d'images apprises est limité à 2 500 bien que le jeu de données d'apprentissage contienne 60 000 images.

Moins les images sont quantifiées, et plus le réseau est en mesure de stocker efficacement les messages. En effet, le nombre de neurones par cluster dépend de la valeur du critère de quantification. Par conséquent, plus le nombre de neurones  $l$  dans un cluster est élevé, et plus faible est la probabilité d’apparition d’une ambiguïté. En effet, un faible nombre de neurones par cluster restreint l’information, et réduit la distinction des cliques durant la phase de récupération.

La densité maximale dans le réseau plein est atteinte avec une architecture de réseau composée de  $l = 10$  neurones plutôt qu’avec  $l = 20$ . Nous pouvons observer que la moyenne des RMSE est meilleure avec SoSP jusqu’à un certain point où les deux stratégies amènent au même résultat. En ce qui concerne le réseau parcimonieux, la variété des données réduit l’apparition des ambiguïtés. Par conséquent, quand le réseau plein est incapable de gérer davantage d’images, le réseau parcimonieux reste efficace comme illustré par la figure 4.10a. Cette dernière fournit une moyenne de RMSE pour des images partiellement effacées reconstruites en fonction du paramètre de pénalisation  $\beta$  pour 30 images apprises avec un CbNN contenant  $l = 20$  neurones par cluster.

Les figures 4.10b et 4.10c illustrent la reconstruction d’images apprises avec et sans pénalisation (pour un réseau plein et parcimonieux respectivement). La dégradation par effacement correspond à trouser l’image en son milieu afin de perdre de l’information que nous cherchons ensuite à reconstruire à l’aide du CbNN. Le réseau initial n’est pas en mesure de corriger l’information, tandis que la pénalisation en fonction d’un  $\beta$  proche de 0 converge vers une clique.

Bien que la méthode proposée permette de repousser les limites en termes de nombre de messages stockés pour une faible erreur de correction durant la phase de récupération, elle amène un coût de calcul supplémentaire pour le terme de pénalisation dans la règle dynamique. Cependant, ce supplément est compensé par des convergences plus rapides.

#### 4.2.3.3 Conclusion

Le réseau à clique est une mémoire associative dotée d’une capacité de récupération intéressante. Cependant, quand les données ne sont pas uniformes, comme c’est le cas avec des images, le réseau fait rapidement face à des problèmes d’ambiguïté, et par conséquent ne réussit pas à récupérer l’information apprise. Nous résolvons ce problème d’ambiguïté en forçant les scores parmi les neurones d’un même cluster à se dissocier à l’aide d’une pénalisation s’appuyant sur la probabilité qu’un neurone appartienne à un message appris (une clique). Les résultats des expériences conduites sur le jeu de données MNIST montrent que la stratégie proposée réduit grandement l’ambiguïté, et par conséquent améliore la reconstruction d’images partiellement effacées (figures 4.10b et 4.10c).

L’apprentissage d’images avec le CbNN incite à explorer une nouvelle famille d’applica-

Méthode		SoS	SoSP	SoS	SoSP	SoS	SoSP	SoS	SoSP	SoS	SoSP
Type	# Images	30		648		1 265		1 883		2 500	
	Quantification										
Plein	$l=10$	35,4	0,0	49,4	38,4	46,3	46,3	47,3	47,3	48,3	48,3
	$l=20$	27,4	26,5	49,6	2,7	46,5	23,9	48,5	47,5	48,3	48,3
Parcimonieux	$l=10$	25,0	4,1	57,6	45,1	51,4	46,0	52,7	49,5	53,0	49,5
	$l=20$	12,5	4,1	47,1	30,2	52,2	48,9	51,1	43,3	51,1	45,9

TABLEAU 4.2 – Moyennes des RMSE sur les images MNIST reconstruites par le CbNN.

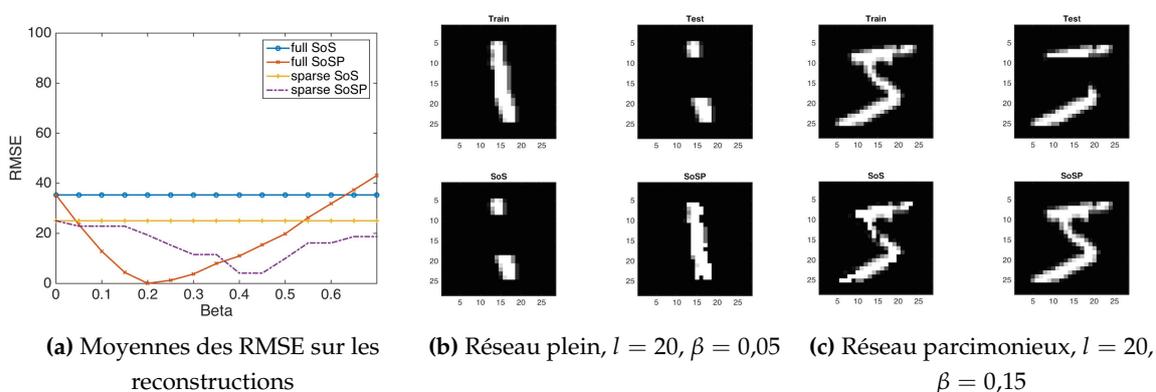


FIGURE 4.10 – Impact du paramètre de pénalisation  $\beta$  sur la reconstruction d'images (a) avec un exemple de reconstruction pour un réseau plein (b) et parcimonieux (c).

tions, la classification.

### 4.3 Classification

Le réseau à clique CbNN peut utiliser sa capacité de correction afin de classer des données apprises au préalable. Ce type d'application est largement utilisé en vision par ordinateur. De manière générale, le but d'une classification discriminante cherche à délimiter les données de nature différentes, autrement dit à catégoriser les données. Cette distinction est définie comme une fonction qui est recherchée par les classifieurs. Cette recherche s'effectue de diverses manières, telles qu'une méthode à noyaux comme avec les machines à vecteurs de support (SVM) [130] ou une régression logistique à travers la méthode de Softmax [17] qui est une généralisation pour un usage multi-classes.

Le but d'un SVM est de classer des données et donc de trouver la frontière permettant de catégoriser celles-ci. La frontière est une combinaison linéaire d'un facteur  $\delta$  et du produit scalaire des données. Le produit scalaire est vu telle qu'un noyau, avec respect de certaines

propriétés. Par conséquent, le SVM recherche le paramètre définissant le facteur  $\delta$  qui maximise la distance, *c.-à-d.* une mesure de similarité comme un noyau calculé entre les données. Parfois les données sont linéairement non séparables, entraînant l'usage d'un autre mode de calcul de similarité entre les données, et nécessitant l'application d'un noyau gaussien par exemple.

Avec l'apparition de l'apprentissage profond (ou *deep learning*), la classification réalisée par ordinateur a dépassé les performances de l'humain sur des tâches spécifiques. En effet, l'élément important dans la classification n'est pas le classifieur lui-même, mais les données soumises à celui-ci [80]. L'utilisation d'un réseau de neurones permet l'extraction de données maximisant la dissociation entre les données. Autrement dit, les réseaux de neurones effectuent un changement de représentation, tout comme l'analyse en composantes principales (ACP), afin d'aider le classifieur à trouver la frontière entre les données.

Pour la classification d'une image, nous souhaitons retrouver l'étiquette associée à celle-ci. Par exemple avec le jeu de données MNIST, la classification permet d'explicitement le chiffre que l'image représente. Ce procédé peut être réalisé de diverses manières. L'apprentissage supervisé s'appuie sur des données connues au préalable pour déterminer les frontières de généralisation afin de classer durant la phase de test des images inconnues. L'apprentissage non supervisé s'appuie sur des méthodes permettant de créer ces frontières, toujours sur la base d'apprentissage mais sans que l'utilisateur ne donne d'étiquettes, la frontière de décision étant alors construite directement à partir de la distribution des données. La stratégie semi-supervisée utilise à la fois les données étiquetées et non étiquetées.

Les meilleures performances de classification sont obtenues par des méthodes qui effectuent en premier lieu une décomposition de l'information. Le but de cette décomposition est d'extraire les caractéristiques des images en fonction des classes auxquelles elles sont associées si la méthode est supervisée. Dans le cas non supervisé, elle recherche une autre forme de caractéristiques pour dissocier les objets les uns des autres. La classification d'images par réseau de neurones sous forme de mémoires associatives n'est pas aussi répandue que les méthodes avec des réseaux de neurones optimisés ou les méthodes à noyaux. Cependant, le *Deep Belief Net* (DBN) [61], comme expliqué dans le chapitre précédent, est un empilement de machines de Boltzmann restreintes (RBM) avec à son sommet une mémoire associative bidirectionnelle. Cette dernière associe les caractéristiques extraites par les RBM aux étiquettes de manière supervisée. Le CbNN a aussi été utilisé sous forme hétéro-associative pour des tâches de classification en se basant sur des extractions de descripteurs locaux tels que des SIFT et des GIST [49].

Le problème de ces méthodes d'extraction est leur caractère unidirectionnel, puisque qu'il est souvent impossible de récupérer l'information initiale après la réduction des données. De plus, les performances de la classification restent optimales si et seulement si les images ne sont pas dégradées. Une méthode permettant de classer des données dégradées

repose sur l'utilisation des auto-encodeurs débruiteurs (ou *Denoising autoencoders*, DA) [132] [133]. Ces réseaux de neurones apprennent la dégradation afin de reconstruire l'information à partir des informations apprises. Cependant, ces réseaux restent de type optimisé, et nécessitent l'usage de l'algorithme de rétro-propagation afin de corriger le gradient des poids de leurs connexions. A l'aide du CbNN nous souhaitons, à l'instar des DA, reconstruire des informations à partir de celles stockées au sein de la mémoire. En d'autres termes, nous souhaitons utiliser le CbNN pour corriger des images dégradées afin de pouvoir ensuite les classer. Cependant, dans cette partie, la classification repose sur les mécaniques propres du CbNN, et non pas sur un classifieur particulier. Par conséquent, le CbNN, en plus de corriger l'information, devient un classifieur. Le protocole d'expérimentation est résumé dans la figure 4.12b.

Les images utilisées restent celles du jeu de données MNIST, dont nous utiliserons les deux ensembles (base d'apprentissage et de test). Le but de nos expérimentations est d'utiliser le CbNN pour classer des données inconnues à l'aide d'un apprentissage supervisé.

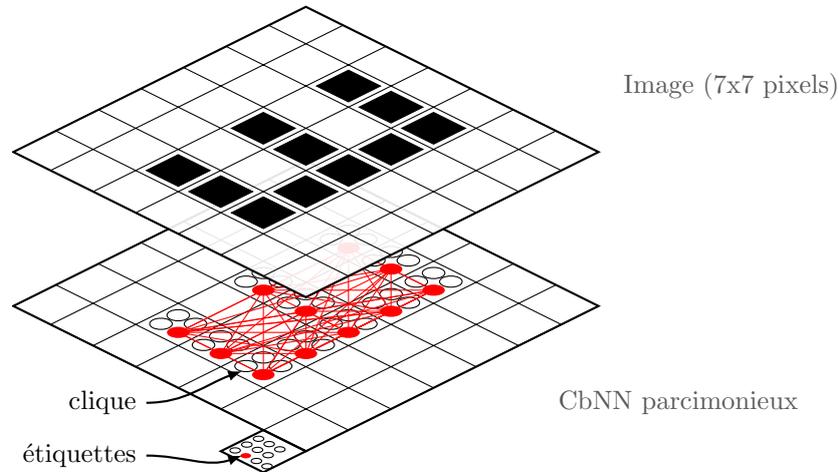
Avec le modèle à couches multiples, la seconde couche était utilisée pour soulager la première (par transfert de cliques) et réduire la densité de celle-ci. Cependant, la seconde couche peut aussi coder une information complémentaire à celles apprises par la première. En effet, la seconde couche peut stocker la classe de l'objet appris par la première couche du réseau. Toutefois, le changement d'architecture dans le cas de la classification entraîne une différence de dimension entre les couches. En revanche, contrairement au modèle à couches multiples initial, les neurones des différentes couches sont connectés. L'apprentissage d'un objet et de sa classe s'effectue par concaténation de ces propriétés. Cette méthode correspond à l'hétéro-associativité qui a été utilisée avec succès sur des descripteurs extraits associés à des étiquettes pour la classification d'images [49].

Dans notre cas, nous souhaitons conserver le caractère auto-associatif du réseau en considérant la couche supplémentaire liée aux étiquettes comme un cluster additionnel de dimension égale au nombre de classes considérées. Par conséquent, la clique encodée comprend à la fois l'information spatiale contenue dans les pixels et la classe de l'image.

### 4.3.1 Méthode

Nous souhaitons utiliser les mécaniques internes du CbNN afin de classer des images issues du jeu de données MNIST. C'est un ensemble de 60 000 images d'entraînement et 10 000 de test. Ce sont des images de 8 bits avec une résolution de  $28 \times 28$  pixels et qui représentent les 10 chiffres manuscrits de 0 à 9.

La première partie de l'architecture dépend des caractéristiques des images. Nous associons un pixel à un cluster,  $\chi = 28 \times 28 = 784$ . Ces clusters sont composés de  $l = 16$  neurones qui codent la valeur des pixels quantifiée par la méthode des K-moyennes sur l'ensemble des



**FIGURE 4.11** – Encodage de l’image d’un 3 dans un réseau à clique CbNN.

images que le réseau va apprendre. Autrement dit, nous quantifions les images en passant de 8 bits à 4 bits. Puis la seconde partie du réseau consiste en un cluster supplémentaire, soit un nombre total  $\chi = 785$ . Ce dernier cluster comporte  $l = 10$  neurones associés aux classes du jeu de données MNIST. La figure 4.11 illustre un exemple de l’apprentissage supervisé d’images par un CbNN avec un cluster supplémentaire lié à la classe pour une tâche de classification.

Le réseau code une image sous la forme d’une clique à travers l’activation de neurones associés aux pixels et à l’étiquette. Dans un souci de lisibilité, les connexions entre le cluster des étiquettes et les autres clusters ont été retirées dans la figure 4.11. La phase d’apprentissage consiste en la transformation des images soumises au réseau en cliques en vue de leur intégration dans la mémoire. L’image et son étiquette sont associées au sein d’une seule clique. Le but est de classer les données en utilisant une règle dynamique afin de calculer les scores des neurones du cluster lié à l’étiquette, et ainsi de sélectionner à l’aide de la règle d’activation WTA la classe associée à l’image soumise. Si l’image a été apprise mais dégradée en entrée, le réseau ne classe pas seulement l’image mais la reconstruit en parallèle. En effet, la phase de récupération recherche la clique entière et par conséquent l’étiquette associée.

Nous utilisons une architecture parcimonieuse du réseau afin d’optimiser le stockage des images. En effet, le fond des images ne représente aucune information, les cliques encodent uniquement les pixels relatifs aux chiffres manuscrits. Par ailleurs, le nombre de clusters actifs n’est pas fixé et dépend de l’image. Par conséquent, deux images différentes peuvent avoir un nombre de clusters actifs différents. Ce dernier point aide à caractériser la diversité des classes. En effet, le chiffre 1 peut contenir moins de pixels blancs qu’un 3 conduisant à une dissociation des cliques en fonction de la classe.

La phase de récupération consiste en l’utilisation alternée d’une règle dynamique (en

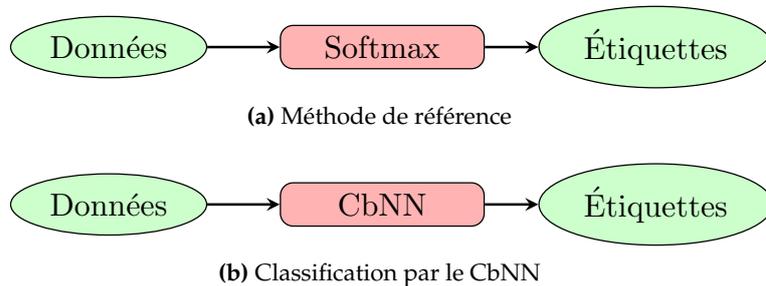


FIGURE 4.12 – Diagrammes des différents types de classification.

l'occurrence le SoS) et d'une règle d'activation. Cette dernière doit être adaptée au type d'application que l'on souhaite utiliser. En effet, dans le cas de la classification, nous souhaitons l'appliquer sur des données inconnues du réseau. Cependant, ce dernier est incapable de généraliser l'information. Par conséquent, l'utilisation de la règle d'activation de type GkWTA est prohibée puisqu'elle ne sélectionnerait que les neurones possédant un score élevé. Cette sélection entraînerait un seuillage de la nouvelle information résultant en une désactivation de la totalité des neurones dans le cluster étiquette. En conclusion, pour les expérimentations, nous utiliserons la règle GkLKO avec  $k$  défini non pas comme le nombre de clusters du réseau, mais fixé empiriquement à  $k = 100$ . Le critère d'arrêt est un nombre maximal d'itérations, sauf si la règle GkLKO désactive l'ensemble restant des neurones. Le nombre maximal d'itérations est fixé à 75.

A l'issue de la phase de récupération, la classe de l'image est définie par le neurone possédant le score le plus élevé au sein du cluster lié à l'étiquette. En effet, l'utilisation de la règle GkLKO désactive les neurones n'ayant pas de lien avec l'entrée soumise à classification. Ces neurones désactivés faisant partie d'une clique voient aussi leur connexion au cluster étiquette se désactiver. Autrement dit, les scores des neurones du cluster associé aux étiquettes évoluent en fonction de la neutralisation des cliques d'une autre classe afin de voir émerger une majorité de cliques d'une classe similaire à l'entrée.

La figure 4.12 présente les différents types de classification que nous avons considérés dans nos travaux. Dans cette partie, nous utilisons le protocole d'expérimentation présenté dans le diagramme 4.12b (le CbNN est un classifieur) et nous considérons celui du diagramme 4.12a à titre de comparaison.

Le classifieur Softmax consiste en une régression logistique multinomiale (linéaire). Il donne pour un exemple la probabilité d'appartenir à une classe. La distribution de probabi-

lité obtenue pour le  $i^{\text{eme}}$  exemple est définie comme

$$h_{\theta}(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1 | x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k | x^{(i)}; \theta) \end{bmatrix}, \quad (4.5)$$

avec  $k$  le nombre de classes (10 dans le cas de MNIST). Nous fixons  $p_j = p(y^{(i)} = j | x^{(i)}; \theta)$  comme la probabilité qu'à l'exemple  $x^{(i)}$  d'être classé comme  $j$  étant donnés les poids  $\theta$ . Cette probabilité est définie par l'équation suivante :

$$p_j = \frac{\exp^{\theta_j^T x}}{\sum_{l=1}^k \exp^{\theta_l^T x}}. \quad (4.6)$$

Dans cette forme, nous obtenons une distribution de  $k$  probabilités (leurs sommes sont égales à 1). La dérivée partielle par  $\theta$  est

$$\frac{\partial p_j}{\partial \theta_i} = \begin{cases} p_i(1 - p_i), & \text{if } i = j \\ -p_i p_j, & \text{if } i \neq j. \end{cases} \quad (4.7)$$

Puisque le Softmax est une méthode supervisée, la fonction de coût nécessite les étiquettes. La fonction de coût entropique croisée pour un exemple est définie comme

$$L = - \sum_j y_j \log(p_j). \quad (4.8)$$

Nous trouvons sa dérivée partielle par  $\theta$  avec les étapes suivantes,

$$\begin{aligned} \frac{\partial L}{\partial \theta_i} &= - \sum_k y_k \frac{\partial \log(p_k)}{\partial \theta_i} \\ &= - \sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial \theta_i} \\ &= -y_i(1 - p_i) - \sum_{k \neq i} y_k \frac{1}{p_k} (-p_k p_i) \\ &= -y_i(1 - p_i) + \sum_{k \neq i} y_k p_i \\ &= -y_i + p_i \sum_k y_k \\ &= p_i - y_i. \end{aligned} \quad (4.9)$$

Le terme  $\sum_k y_k$  est égal à 1 puisque  $y$  est un vecteur avec seulement 1 élément non nul égal à 1, à la position correspondant à l'étiquette associée. Si nous généralisons la fonction de coût à  $m$  exemples et ajoutons un terme de régularisation avec un paramètre de décroissance de poids  $\lambda$  afin de prévenir un sur-apprentissage, nous obtenons

$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{j=1}^k \mathbb{1}_{\{y^{(i)}=j\}} \log(p_j) \right] + \frac{\lambda}{2} \sum_{i=1}^k \sum_{j=0}^n \theta_{ij}^2. \quad (4.10)$$

La dérivée partielle par le paramètre  $\theta$  en utilisant l'équation (4.9) est

$$\nabla_{\theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m x^{(i)} \left( p_j - \mathbb{1}_{\{y^{(i)}=j\}} \right) + \lambda \theta_j, \quad (4.11)$$

avec  $\mathbb{1}_{\{y^{(i)}=j\}}$  une matrice parcimonieuse qui encode l'appartenance aux classes. Le Softmax effectue une régression afin de mettre sous forme probabilisée l'appartenance à une classe pour une image donnée. Cette régression est obtenue à travers une optimisation convexe sur l'ensemble des données d'apprentissage. La prédiction de classe sur le jeu de test est réalisée à l'aide d'un produit matriciel avec la matrice de poids correspondant à la régression apprise durant la phase d'entraînement.

L'approche suivie par le CbNN est différente. En effet, la phase d'apprentissage correspond à l'activation de neurones en fonction des données fournies, et ce de manière indépendante pour chacune d'elle. Quand le Softmax cherche à départager les images de manière supervisé et effective, le CbNN se limite à enregistrer les données dans sa mémoire. Autrement dit, il est moins actif que le classifieur. C'est la même distinction qu'on peut observer entre les réseaux de neurones optimisés par l'algorithme de rétro-propagation ou de divergence contrastive et les mémoires associatives expliquées plus en détail dans le chapitre précédent.

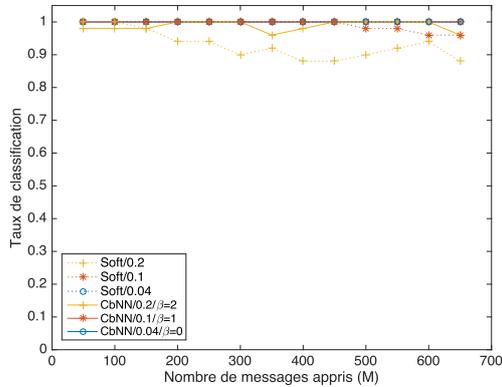
Durant la phase de test, le classifieur Softmax consiste en un produit matriciel, alors que la classification réalisée par le CbNN consiste à converger itérativement vers une clique proche de l'image donnée, afin de récupérer la classe de cette dernière.

### 4.3.2 Expérimentations

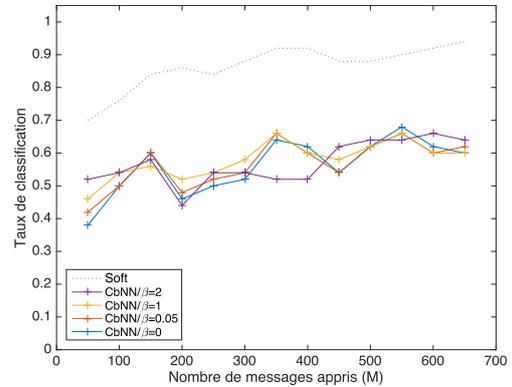
Les résultats des expérimentations sont issus de trois scénarios de classification : (1) images d'entraînement dégradées, (2) images de test (non apprises sans dégradation), et enfin (3) images de test dégradées.

Les dégradations possibles sont au nombre de trois. La première consiste en une altération poivre et sel (visible dans la figure 4.13f, en haut à gauche) en fonction d'un facteur de dégradation graduel. Le deuxième type correspond à l'effacement de pixels déjà utilisé pour la pénalisation (figure 4.10). L'effacement des pixels n'est plus ici réalisé dans un unique bloc, mais correspond à une suppression aléatoire des pixels afin de dégrader l'information effective suivant un facteur de dégradation. Le dernier type de dégradation prend la forme d'ajout de bruit gaussien selon différents facteurs d'altération. Outre ces facteurs, nous tenons compte du paramètre de pénalisation  $\beta$  pour identifier son influence dans la tâche de classification. Il est à noter que  $\beta = 0$  correspond au CbNN sans pénalisation.

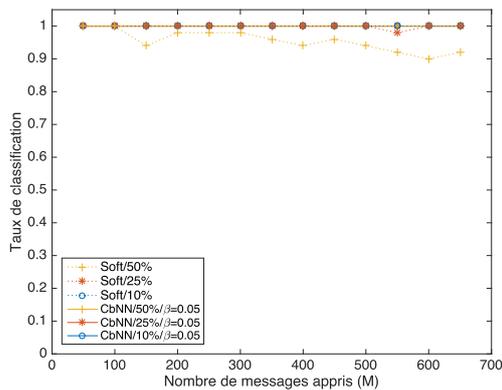
Les résultats de classification nous permettent de comparer les performances du CbNN au Softmax. Ce dernier est entraîné exactement sur les mêmes données que le CbNN (les images d'entraînement du Softmax sont ainsi quantifiées). La classification est réalisée sur



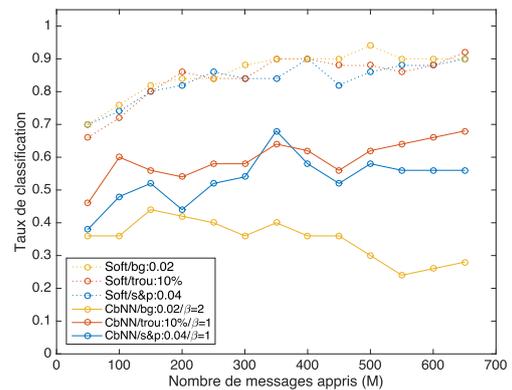
(a) Images : entraînement /  
dégradation : poivre et sel



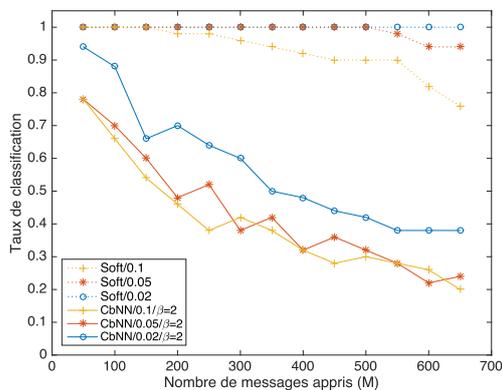
(b) Images : test / dégradation : aucune



(c) Images : entraînement /  
dégradation : trous aléatoires



(d) Images : test / dégradation : toutes



(e) Images : entraînement /  
dégradation : bruit gaussien



(f) Reconstructions (à gauche : images  
soumises ; à droite : images récupérées)

**FIGURE 4.13** – Taux de classification du CbNN et d'un Softmax sur des images non apprises en fonction du nombre de messages appris, et reconstructions illustrant le type de dégradation appliquée.

des images de test préalablement quantifiées. La dégradation, si elle a lieu, est appliquée avant quantification. Les résultats de la figure 4.13 sont les moyennes des classifications sur les mêmes 50 premières images d'entraînement ou de test en fonction du type d'évaluation. A l'exception de la figure 4.13b, toutes les figures illustrent les meilleurs résultats avec pénalisation (donc le  $\beta$  n'est pas constant).

Comme l'illustrent les figures 4.13a et 4.13c, la classification sur des données apprises avec une dégradation consistant en de l'effacement, le CbNN est capable de reconstruire et de classer parfaitement jusqu'à 650 images apprises. Dans ce type d'évaluation, le CbNN à travers sa capacité correctrice obtient de meilleures performances que le Softmax avec un effacement de 25% et 50% de l'information. En effet, le Softmax est entraîné sur des images non dégradées et est incapable de généralisation dans le cas d'images dégradées. Par contre, comme le montre la figure 4.13e illustrant les résultats de classification avec une dégradation de type bruit additif, le CbNN est incapable de récupérer l'information par manque de généralisation, entraînant une chute des performances. Avec la dégradation par un bruit de type poivre et sel, l'ajout de données est limité et contrebalancé par un effacement permettant au CbNN de récupérer l'information. En revanche, le bruit gaussien entraîne l'activation d'un plus grand ensemble de neurones. Cependant, ces neurones font partie de cliques différentes. Par conséquent, la classification s'appuyant sur le cluster étiquette n'est pas en mesure de dissocier une classe d'une autre, d'où une diminution des performances.

La classification d'images non apprises sans dégradation, illustrée par la figure 4.13b, démontre l'incapacité du réseau à généraliser la classe de l'objet qu'il a appris. L'image de reconstruction de la ligne du milieu correspondant à une image non apprise de la figure 4.13f est vide. En effet, l'utilisation de la règle d'activation GkLKO désactive au fur et mesure les neurones afin de récupérer uniquement ceux en lien avec l'image donnée et obtenir leurs classes associées. Cependant, par faute de convergence, les neurones peuvent se retrouver tous désactivés.

L'étude de la pénalisation était concentrée sur des images apprises. La figure 4.13b illustre l'influence négligeable de la pénalisation sur des images inconnues. En effet, la contrainte de pénalisation maximise l'activation des neurones formant une clique pour représenter l'entrée soumise. Cependant, si l'image soumise est inconnue, la maximisation d'activation ne concerne plus une clique mais un ensemble de cliques conduisant à l'inefficacité de la pénalisation sur des images inconnues.

Le troisième type de test correspond à des images inconnues dégradées et s'appuie sur la capacité du CbNN à reconstruire une information manquante sur la base de ce qu'il a déjà appris. Cependant, le résultat de classification sur ce type de données est similaire à celui obtenu pour des images non dégradées. La figure 4.13d montre que la correction d'effacement est plus efficace que le bruit additif. En effet, pour la reconstruction d'images partielles, le CbNN s'appuie sur ses connaissances en lien avec l'image d'entrée. S'il ne la connaît pas, il

est alors dans l'incapacité de reconstruire l'image. La seule possibilité de succès de reconstruction est observée dans le cas où la dégradation de l'image conduit à une image apprise. La reconstruction de données non apprises et dégradées est illustrée par la dernière ligne de la figure 4.13f. Le réseau CbNN a été en mesure de récupérer des neurones actifs étant donnée l'entrée, mais celle-ci a été classée comme une image de 3 alors que l'étiquette était un 7.

Le changement d'architecture pour utiliser le CbNN comme un classifieur se révèle performant dans le cas d'images apprises si le bruit additif est limité. En effet, les résultats de classification sur des images apprises et dégradées par du bruit poivre et sel ou par un effacement de manière plus générale montrent que le CbNN est en mesure de classer par reconstruction d'information. De plus, sur ce type de dégradation, le CbNN obtient des performances équivalentes ou supérieures à la méthode Softmax.

Que l'objectif soit de corriger ou de classer une information par reconstruction, le réseau CbNN reste incapable de traiter des données non apprises au préalable, dégradées ou non. Par conséquent, nous étudions ensuite une opération supplémentaire aux règles utilisées par le réseau afin d'aider (à l'instar de la pénalisation) le CbNN à généraliser les informations apprises.

## 4.4 Généralisation par a priori

Nous avons discuté dans le chapitre précédent de l'incapacité du CbNN à généraliser l'information apprise au sein sa mémoire. En effet, la soumission au réseau d'un message inconnu lors de la phase de récupération ne lui permet pas de récupérer une clique apprise, car cette phase converge vers un chevauchement de cliques. Ce chevauchement est le résultat de l'activation des neurones possédant des scores élevés au vu de l'entrée soumise, mais appartenant pourtant à des cliques différentes. Par conséquent, le CbNN est incapable de distinguer une clique d'une autre faute d'inférence. Nous proposons une méthode s'appuyant sur la pénalisation afin de favoriser l'activation des neurones proches de ceux qui devraient l'être étant donnée une entrée soumise inconnue.

### 4.4.1 Utilisation d'a priori

Jusqu'à présent tous les résultats présentés démontrent l'inefficacité du réseau CbNN à gérer les données inconnues. Nous proposons ici une méthode s'appuyant sur la pénalisation. La stratégie bayésienne de la méthode cherche à calculer une probabilité d'activation pour un neurone donné en fonction de l'entrée, connue ou non, conditionnellement à ce qui a été appris au préalable par le réseau. A l'instar de la pénalisation, nous souhaitons utiliser un a priori dans le calcul de score des neurones afin de mettre en avant ceux devant être

activés. Le but de cette méthode est de reconstruire une clique apprise à partir d'une entrée inconnue.

Soit  $\phi_i$  l'a priori du cluster  $i$  correspondant au neurone qui devrait être activé en fonction de l'entrée soumise. De même que la pénalisation, l'a priori est un processus qui s'insère entre la règle dynamique et celle d'activation. Une fois les scores  $s_{i,j}$  calculés (équation 4.3), ils sont normalisés afin de les représenter sous forme de distributions de probabilités. Soit  $\mathbb{P}(v_{i,j})$  la probabilité que le neurone  $j$  du cluster  $i$  soit activé

$$\mathbb{P}(v_{i,j}) = \frac{s_{i,j}}{\sum_k s_{k,j}}. \quad (4.12)$$

Afin d'améliorer la sélection de neurones pour un a priori donné par l'entrée soumise, nous calculons la probabilité postérieure de l'activation  $\mathbb{P}(v_{i,j} \mid \phi_i)$  en utilisant le théorème de Bayes sur les probabilités conditionnelles. A cet égard, il faut calculer au préalable la vraisemblance d'activation pour un neurone donné, soit

$$\mathbb{P}(\phi_i \mid v_{i,j}) = \frac{f(|\phi_i - v_{i,j}|)}{\sum_j f(|\phi_i - v_{i,j}|)} \quad (4.13)$$

avec  $f$  la fonction définie par  $f(x) = 1/(1+x)$ . La méthode d'a priori donne ainsi plus de poids aux neurones actifs les plus proches de l'a priori soumis en entrée.

La probabilité de l'a priori est calculée à partir des probabilités totales, et elle est définie comme

$$\mathbb{P}(\phi_i) = \sum_j \mathbb{P}(\phi_i \mid v_{i,j})\mathbb{P}(v_{i,j}). \quad (4.14)$$

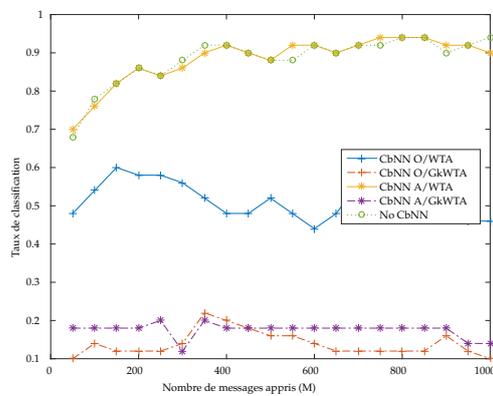
La probabilité d'activation d'un neurone en fonction d'un a priori est finalement obtenue avec le théorème de Bayes,

$$\mathbb{P}(v_{i,j} \mid \phi_i) = \frac{\mathbb{P}(\phi_i \mid v_{i,j})\mathbb{P}(v_{i,j})}{\mathbb{P}(\phi_i)}. \quad (4.15)$$

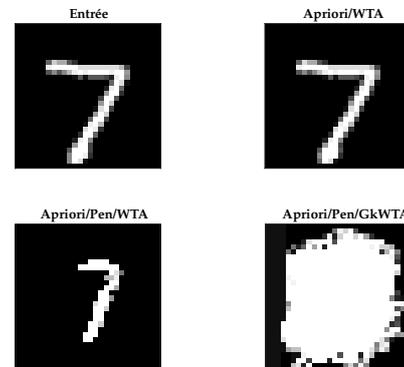
L'effet de cette méthode, postérieure à la règle dynamique, est analysé dans la partie suivante, à travers plusieurs expérimentations.

#### 4.4.2 Expérimentations

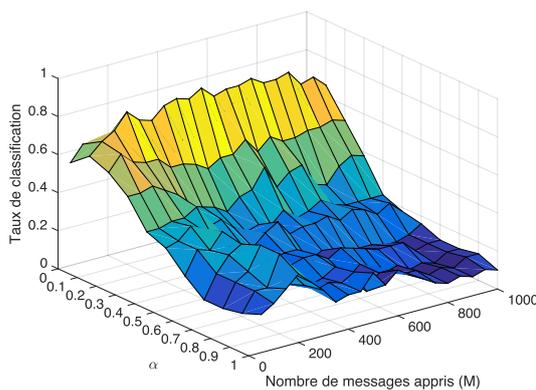
Les expérimentations se focalisent ici sur des images non apprises par le réseau CbNN. Afin de mesurer la capacité de généralisation du réseau, nous utilisons la méthode Softmax pour classer les images quantifiées. Puis les images sont corrigées par le CbNN après un apprentissage sur la base d'entraînement (quantifiée au préalable). Le processus suit le diagramme de la figure 4.15.



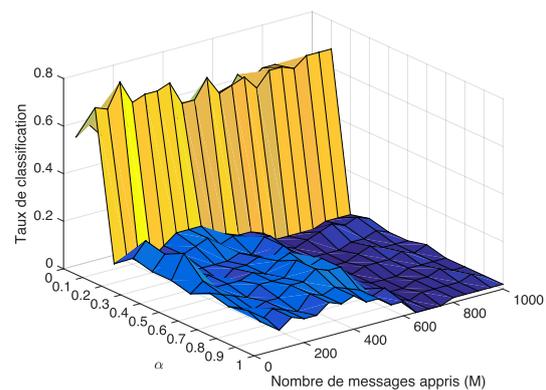
(a) Taux de classification pour différentes architectures de CbNN



(b) Reconstructions



(c) Taux de classification avec pénalisation sur un CbNN plein



(d) Taux de classification avec pénalisation sur un CbNN parcimonieux

FIGURE 4.14 – Résultats d'utilisation d'a priori sur des images inconnues au réseau.

La première partie des expérimentations consiste à étudier l'influence de l'a priori sur la reconstruction des images inconnues du réseau. Pour ce faire, le réseau apprend de manière incrémentale jusqu'à 1 000 images. Toutes les 50 images, nous reconstruisons 50 images non apprises par le réseau qui restent identiques pour chaque test. Deux architectures sont utilisées : la première est pleine et implique l'usage de la règle d'activation WTA ; la seconde est parcimonieuse et utilise la règle GkWTA et un nombre de clusters actif passant de  $c = \chi = 784$  à  $c = 166$ . Le nombre de clusters dépend du nombre de pixels dans l'image. Chaque cluster comprend  $l = 16$  neurones. La quantification est effectuée par la méthode des K-moyennes appliquée à l'ensemble des 1 000 images. Les deux architectures utilisent la règle dynamique SoS.

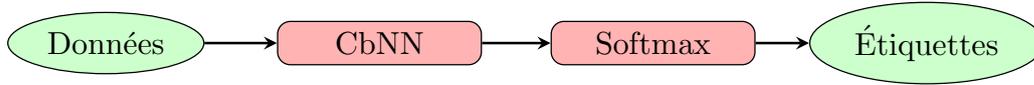


FIGURE 4.15 – Diagramme de classification pour le CbNN.

Les résultats de classification suivent le diagramme 4.15 (le Softmax est le classifieur des données corrigées par le CbNN). La figure 4.14a illustre le taux de classification s'appuyant sur l'analyse des pixels après reconstruction des images non apprises à l'aide de la méthode Softmax (figure 4.12a, courbe No CbNN).

La méthode Softmax sans correction du CbNN sert de référence et tend vers 1, indiquant que plus la méthode apprend d'images et mieux elle généralise l'information. L'usage de l'a priori avec un réseau plein (CbNN A/WTA) obtient approximativement les mêmes résultats que le Softmax. En effet, avec la règle locale WTA, les neurones tendent à être activés en fonction de l'a priori de l'image soumise. Le problème vient du fait que le WTA n'induit pas une convergence vers une clique, mais consiste en l'activation des neurones associés à l'entrée, comme en témoigne un exemple de reconstruction de la figure 4.14b dans le coin en haut à droite. Le CbNN plein original (CbNN O/WTA) obtient des performances avoisinant 50% de succès de classification du fait de la redondance d'information sur l'ensemble des connexions neuronales.

Le passage à une architecture parcimonieuse dégrade les performances, et même l'utilisation de l'a priori ne peut aider le réseau à corriger l'information. En effet, l'a priori concentre trop fortement l'activation d'un neurone précis, conduisant l'activation d'un neurone qui ne devrait pas l'être. Par conséquent, la règle GkWTA active un ensemble de neurones n'appartenant pas à la même clique, entraînant alors un chevauchement de cliques, et donc une image illisible comme présentée dans la figure 3.11.

Dans le but de réduire l'effet de l'a priori et de contraindre la convergence vers une clique apprise, nous utilisons la pénalisation afin de contrebalancer l'a priori qui résulte d'une approximation locale sans prendre en compte l'ensemble des possibilités du réseau. Pour un neurone donné, le nouveau score obtenu avec l'a priori (équation 4.15) et la pénalisation (équation 4.4) est défini par

$$\lambda_{i,j} = (1 - \alpha)\mathbb{P}(v_{i,j} | \phi_i) + \alpha\hat{\lambda}_{i,j}, \quad (4.16)$$

avec  $v_{i,j}$  le neurone  $j$  du cluster  $i$ .

Les figures 4.14c et 4.14d présentent le taux de classification en fonction du paramètre  $\alpha$ , pour un réseau plein et parcimonieux respectivement. Le paramètre gérant l'effet de pénalisation est fixé à sa valeur optimale déterminée empiriquement, soit  $\beta = 0,05$ . Nous souhaitons utiliser la pénalisation afin de contraindre les activations à converger vers une clique apprise, comme l'illustre la figure de reconstruction 4.14b dans le coin en bas à gauche. En

effet, pour  $\alpha = 0,1$  (faible effet de l'a priori), la reconstruction montre que le réseau ne produit plus l'image d'entrée mais une approximation qui n'est toujours pas une clique. Quand le paramètre  $\alpha$  tend vers 1, impliquant une utilisation exclusive de la pénalisation, les performances de généralisation sont au plus bas, illustrant l'impossibilité du CbNN à converger, même sous contrainte, vers une clique apprise. En ce qui concerne le réseau parcimonieux, les résultats de classification sont proches avec ou sans pénalisation. Par conséquent, la production d'un chevauchement de cliques à partir d'images inconnues n'est pas inhérente aux contraintes ajoutées au réseau, mais bien aux mécaniques propres au CbNN qui empêchent la gestion de données non apprises au préalable.

Nous pouvons conclure de ces expériences que le problème de généralisation provient des mécaniques du CbNN et que même une modification de ces dernières ne permet pas d'améliorer la phase de récupération. Il nous semble par conséquent intéressant de changer la structure même du réseau CbNN, ce que nous étudierons dans le prochain chapitre.

## 4.5 Conclusion

Le réseau de neurones à cliques possède deux principaux défauts. Le premier concerne la gestion de données non indépendantes et identiquement distribuées telles que des images, qui induisent des ambiguïtés pendant la phase de récupération. Une ambiguïté apparaît lorsque le réseau CbNN n'est plus en mesure de sélectionner le neurone adéquat à activer pour représenter l'information à reconstruire. Ce phénomène d'ambiguïté s'explique par la nature binaire du réseau. En effet, une fois un neurone activé pour représenter une clique, il restera activé dans le réseau jusqu'au remplissage complet de la matrice d'adjacence mélangeant la totalité des cliques apprises. Par conséquent, la binarité du réseau limite la dissociation des cliques lors de la récupération d'information, conduisant ainsi à des ambiguïtés.

Nous avons proposé un réseau CbNN composé de plusieurs couches afin de mieux répartir les cliques. En réduisant les ambiguïtés, la performance est accrue. Cependant, cela peut nécessiter une optimisation sous forme d'une factorisation de la matrice d'adjacence si l'on souhaite éviter le surcoût de mémoire généré par le stockage complémentaire des cliques. La nouvelle plasticité a démontré que la densité d'un CbNN est un facteur déterminant dans la capacité correctrice du réseau.

Afin de conserver l'architecture initiale, nous avons également proposé une méthode de pénalisation appliquée après la règle dynamique. Elle a pour effet de réduire l'ambiguïté sur tous les types de données apprises au préalable (y compris les images). En effet, la clique est un graphe complet où les degrés des neurones connectés sont tous identiques. La pénalisation exploite cette propriété afin de reconstruire la clique. Notons que si les images n'ont pas été préalablement apprises, la pénalisation ne pourra pas aider le réseau à converger vers une clique apprise.

Le réseau CbNN n'est pas limité à la reconstruction d'image. En effet, sa capacité de correction lui permet aussi de les classer. Pour cela, on propose de modifier son architecture en ajoutant un cluster supplémentaire qui enregistre la classe de l'image apprise. Les performances de classification sur des images dégradées qui sont préalablement apprises sont supérieures à celles du Softmax. En revanche, lorsque les images sont inconnues, le CbNN fournit de moins bons résultats que le Softmax. Plus précisément, le réseau est capable de gérer correctement des dégradations de type effacement, mais beaucoup moins de corriger des données bruitées. En effet, le bruit se traduit par des informations inconnues du réseau. Plus généralement, le second problème rencontré par le CbNN est la généralisation.

En s'appuyant sur le principe de pénalisation, nous avons abordé ce problème en proposant une méthode d'a priori pour guider la récupération vers une clique apprise. Cependant, le caractère purement local de la récupération est accru par ces a priori, car ils n'opèrent que sur les neurones et non sur les clusters. Même en intégrant une étape de pénalisation, le réseau reste incapable de retrouver une entrée inconnue. Puisque les modifications apportées ici au réseau (a priori, a priori avec pénalisation) ne permettent pas de généraliser, nous proposons dans le prochain chapitre une architecture originale de mémoire associative a clique bidirectionnelle. Celle-ci permettra d'appréhender globalement l'information fournie au réseau.

# Chapitre 5

## Mémoire associative bidirectionnelle à base de clique

---

### Contenu

---

<b>5.1</b>	<b>Introduction</b>	<b>92</b>
<b>5.2</b>	<b>Concept du modèle BCAM</b>	<b>93</b>
5.2.1	Phase d'apprentissage	96
5.2.2	Phase de récupération	97
<b>5.3</b>	<b>Évolution du modèle</b>	<b>100</b>
5.3.1	Méthode d'acquisition comprimée	100
5.3.2	Topologie adaptative	103
5.3.3	Variantes du modèle BCAM	107
5.3.3.1	BCAM-BAM	107
5.3.3.2	BCAM-CS	107
<b>5.4</b>	<b>Expérimentations</b>	<b>108</b>
5.4.1	Données générées artificiellement	111
5.4.2	MNIST	114
5.4.3	Autres jeux d'images	118
<b>5.5</b>	<b>Discussion sur la généralisation</b>	<b>122</b>
5.5.1	Étude de la dimension des patches	126
<b>5.6</b>	<b>Conclusion</b>	<b>127</b>

---

## 5.1 Introduction

Une fonction de la mémoire associative humaine est d'associer de nouvelles entrées à des informations apprises au préalable. Autrement dit, notre mémoire réalise une approximation d'informations inconnues à l'aide de sa connaissance. Ce principe illustre la notion de généralisation. Dans le cas des modèles d'apprentissage automatique ou de mémoire associative neuronale, le but reste le même : nous souhaitons que les modèles puissent appréhender l'inconnu sur la base des informations dont ils ont déjà connaissance. Les mémoires associatives de Hopfield sont capables de généralisation à travers l'optimisation de la fonction de coût associée au réseau. L'optimisation permet, à partir d'une entrée inconnue, de converger presque sûrement vers ce qui a été appris, notamment à l'aide des bassins d'attraction générés par la fonction d'énergie liée à l'équilibre du réseau. Contrairement aux modèles de Hopfield, la phase de récupération du CbNN ne s'appuie pas sur l'optimisation d'une fonction de coût. En effet, la reconstruction est réalisée à l'aide d'informations communes entre celles stockées au sein de la mémoire et celles soumises. Dans le cas du CbNN, il y a une notion d'inférence.

À travers les chapitres précédents présentant le CbNN et ses mécaniques de récupération, nous avons souligné l'incapacité du réseau à corriger des informations inconnues à cause de l'apparition des chevauchements de cliques. En effet, la phase de correction correspond à l'activation de neurones ne possédant aucun a priori global sur l'entrée soumise. Autrement dit, si une entrée non apprise au préalable est soumise au réseau, les neurones associés durant la phase de récupération seront activés sans prendre en considération leur appartenance à une seule et unique clique. Par conséquent, les neurones s'activent non pas pour une clique, mais pour un chevauchement de sous-cliques. Une solution pour éviter l'apparition d'un chevauchement de cliques consiste à ajouter, au moment de la phase de récupération, une contrainte d'appartenance à une unique clique à l'aide de la pénalisation présentée dans le chapitre précédent. En effet, les résultats de récupération à l'aide de la pénalisation montrent une amélioration de la désambiguïsation des cliques, limitant les chevauchements plus fréquents avec des données non uniformes. Cependant, en plus de la nécessité de réaliser une validation croisée du paramètre de pénalisation, nous avons aussi démontré l'incapacité de ce dernier à gérer les données inconnues. En effet, le paramètre de pénalisation, à l'instar de celui de l'a priori, est appliqué directement sur les neurones, *c.-à-d.* localement. Ainsi, la pénalisation permet la désambiguïsation à faible échelle mais, dans le cas d'information inconnues, les ambiguïtés ne sont pas localisées sur un sous-ensemble de neurones, mais sur l'ensemble de ces derniers.

Les contributions détaillées dans le chapitre précédent pour aider le CbNN à réaliser une généralisation ont porté sur la modification des mécaniques internes du réseau. Cependant, nous avons démontré que de telles modifications ne permettent toujours pas au réseau de

généraliser les informations contenues au sein de sa mémoire. La littérature se focalise quant à elle sur la correction de données artificielles de faible dimension dans le but de réduire les ambiguïtés dans le réseau, et non pas sur la notion même de généralisation. Cependant, ces contributions sur les ambiguïtés permettent par la même occasion d’approcher la notion de généralisation. Par exemple, l’utilisation d’étiquettes sur les cliques [91] permet leurs différenciation, créant ainsi une distinction de ces dernières durant la phase de récupération à une échelle plus globale. Dans ce chapitre, nous proposons une contribution reposant sur un changement d’architecture du CbNN. Nous introduisons une nouvelle architecture de CbNN à couches multiples nommée Mémoire Associative à Cliques Bidirectionnelle (ou Bidirectional Cliques based Associative Memory, BCAM) afin d’apporter la notion de généralisation aux réseaux à cliques. Le réseau BCAM est composé de deux couches, et s’appuie sur les résultats de notre étude sur le CbNN composé de plusieurs couches et discuté dans le chapitre précédent. La différence avec le modèle à couches multiples réside dans le passage entre les couches qui s’effectue ici par une matrice d’association bidirectionnelle. Le passage entre les couches repose sur le concept de la mémoire associative bidirectionnelle (BAM) [88] détaillée dans le chapitre dédié à la présentation de l’état de l’art.

La première section présente le modèle à travers son architecture et ses phases d’apprentissage et de récupération. La section suivante se focalise sur son potentiel d’évolution. La troisième section décrit les expérimentations réalisées sur des données artificielles apprises et dégradées afin de démontrer la conservation des capacités des CbNN au sein du BCAM, puis sur des images inconnues afin de valider le modèle compte tenu du critère de généralisation. Nous étudions plus en détail dans la dernière section comment le réseau est capable de réaliser la généralisation.

## 5.2 Concept du modèle BCAM

L’apprentissage d’images avec un CbNN est une problématique forte due à la consommation de la mémoire engendrée par ces images. En effet, comme indiqué dans le chapitre précédent, l’apprentissage d’images s’appuyant sur une reconstruction par le réseau nécessite une architecture adaptée aux dimensions des images. Le nombre de clusters dépend du nombre de pixels dans l’image, et le nombre de neurones par cluster dépend de la profondeur d’un pixel (sa plage de valeurs). À titre d’exemple, pour des images de résolution  $28 \times 28$  et de profondeur 8 bits, le nombre de neurones du réseau serait de  $28^2 \times 2^8 = 200\,704$ . La matrice d’adjacence associée posséderait une dimension de  $200\,704 \times 200\,704$ , portant le nombre d’entrées de la matrice à 40 282 095 616. Un réseau de cette dimension requiert coût mémoire probablement prohibitif, surtout si nous souhaitons transposer le réseau sur une puce à faibles ressources. Une solution utilisée avec succès pour les expérimentations sur les images réalisées dans les sections (4.2.3) et (4.3) afin de réduire la dimension consiste

à réaliser une quantification par la méthode des K-moyennes. En effet, la quantification réduit le nombre de neurones par cluster et donc quadratiquement la dimension de la matrice d'adjacence associée au réseau.

Le besoin en mémoire relatif à l'apprentissage d'images dépend de la matrice d'adjacence qui répertorie l'ensemble des connexions entre les pixels. Les résultats sur le modèle à couches multiples permettent de démontrer qu'une plasticité au sein du CbNN allège la densité des couches utilisées, et par conséquent réduit la consommation mémoire du réseau. Nous nous appuyons sur ces résultats et découpons les images afin d'associer les patches de chacune d'elles à des sous-réseaux CbNN. En effet, les sous-réseaux codent uniquement l'information relatives à leurs patches telle que l'illustre la figure 5.1. Par conséquent, la dimension des matrices d'adjacence associées aux sous-réseaux est réduite, atténuant par la même occasion la consommation mémoire.

La première couche du réseau est composée de sous-réseaux, chacun étant associé à un patch d'une image décomposée. En reprenant l'exemple précédent, nous considérons une image de résolution  $28 \times 28$  et de profondeur 8 bits, et nous la découpons en  $p = 16$  parties sans chevauchement. Par conséquent, la première couche est composée de 16 sous-réseaux CbNN possédant chacun  $\chi = 7 \times 7 = 49$  clusters, correspondant à la résolution des patches associés. Le nombre de neurones par cluster est  $l = 256$  dû au codage 8 bits des pixels. En terme de mémoire, le nombre total d'entrées à travers la concaténation des matrices d'adjacences devient  $(49 \times 256)^2 \times 16 = 2\,517\,630\,976 \ll 40\,282\,095\,616$ . Même si le nombre d'entrées reste élevé, justifiant l'usage de la quantification, le changement d'architecture a néanmoins permis de diviser par 16 l'impact de la mémoire sur cet exemple. Autrement dit, par rapport à la matrice d'adjacence du réseau initial, seuls les blocs diagonaux de dimension égale aux patches sont pris en considération dans la première couche du BCAM. Ce gain de mémoire est compensé par l'ajout d'une couche supplémentaire composée d'un unique CbNN et une matrice de passage entre les deux couches.

En considérant toujours l'exemple précédent, nous calculons les capacités associées au réseau initial et à la première couche du BCAM à l'aide de l'équation (3.4). Nous obtenons une capacité  $Q_{cbnn} = 2,0115e10$  pour le modèle initial, quand celle de la première couche du nouveau réseau est  $Q_{bcam}^{(1)} = 1,2331e09$  pour l'ensemble des sous-réseaux CbNN. Bien que la capacité soit moindre, la particularité du BCAM est sa plasticité à travers l'ajout d'une seconde couche. En effet, quand le réseau initial connecte tous les pixels d'une image entre eux à travers l'activation des neurones associés, le BCAM connecte seulement ceux issus du même patch. C'est la seconde couche qui connecte les sous-réseaux entre eux afin d'associer les patches en une seule image. Par conséquent, la première couche code les informations locales en fonction des pixels, et la seconde couche code l'information globale afin d'exprimer l'appartenance des cliques locales à la même image.

Le nombre de clusters du CbNN de la seconde couche est égal au nombre de sous-

réseaux dans la première. Quant au nombre de neurones, il dépend de l'évolution de la densité de la première couche. En effet, si une couche possède une densité plus élevée que l'autre, elle deviendra le facteur limitant du réseau, résultant alors en une réduction de performances en termes de reconstruction. Le fait que le réseau en seconde couche soit composé d'un faible nombre de clusters n'est pas un désavantage tant que le nombre de neurones qui les composent est ajusté en fonction du volume d'information à stocker dans le réseau. En effet, les expérimentations de la section (3.2.2) démontrent que le nombre de neurones par cluster est plus important que le nombre des clusters. En outre, un réseau possédant peu de clusters, mais un nombre élevé de neurones, est considéré comme optimal si la structure est parcimonieuse, *c.-à-d.* si un sous-ensemble des clusters est actif pour représenter l'information, soit  $c < \chi$ . Afin de calculer le nombre de neurones dans  $L_2$  et d'harmoniser les deux couches, nous nous appuyons sur l'équation (3.6) qui tient compte de la densité. En effet, la densité évolue en fonction du nombre de connexions possibles, ou autrement dit la capacité dans le réseau définie par l'équation (3.4). La capacité dépend de l'architecture du réseau. Par conséquent, nous recherchons  $l_2$  tel que la capacité  $Q^{(1)}$  de la première couche soit égale à celle de la seconde couche  $Q^{(2)}$ . Notons que le raisonnement est valable uniquement dans le cas où les données des deux couches suivent la même distribution. Si c'est le cas, le nombre de neurones dans la seconde couche est défini par l'équation suivante,

$$l_2 = \sqrt{\frac{\chi_1(\chi_1 - 1)l_1^2 p}{\chi_2(\chi_2 - 1)}}. \quad (5.1)$$

Sinon, le choix du nombre de neurones  $l_2$  peut être réalisé arbitrairement sous la condition qu'il ne limite pas la diversité de la seconde couche par rapport à la première. Pour l'exemple d'images de résolution  $28 \times 28$  de profondeur 8 bits découpées en  $p = 16$  patches sans chevauchement, les architectures des deux couches sont détaillées dans le tableau

$p$	$\chi_1$	$l_1$	$\chi_2$	$l_2$
16	49	256	16	3 206

**TABLEAU 5.1** – Architecture d'un BCAM en fonction d'un découpage des données en  $p = 16$  parties.

La clé de voûte du modèle est le passage bidirectionnel de l'information d'une couche à une autre. En effet, la récupération est réalisée itérativement à travers le transit bidirectionnel des informations corrigées par les deux couches. La transition est effectuée par produit matriciel avec la matrice de poids connectant les deux couches. Cette matrice de poids  $Z$  est une résultante du produit matriciel entre les cliques de la première couche et celles de la seconde couche, c'est donc une matrice d'association.

Soit  $x_{i,j}^{(k)} \in \{0;1\}$  la valeur du  $i$ -ième neurone pour le  $j$ -ième message dans le  $k$ -ième CbNN de la première couche,  $k = 1, \dots, p$ ,  $i = 1, \dots, n_1$ ,  $j = 1, \dots, M$ , avec  $p$  le nombre

de sous-réseaux,  $n_1$  le nombre total de neurones par CbNN de la première couche, et  $M$  le nombre de messages. On pose  $X \in \{0;1\}^{n_1 \times p, M}$  la concaténation des  $x^{(k)}$  telle que

$$X = \begin{pmatrix} x_{1,1}^{(1)} & \cdots & x_{1,M}^{(1)} \\ \vdots & x_{i,j}^{(1)} & \vdots \\ x_{n_1,1}^{(1)} & \cdots & x_{n_1,M}^{(1)} \\ \vdots & & \vdots \\ x_{1,1}^{(p)} & \cdots & x_{1,M}^{(p)} \\ \vdots & x_{i,j}^{(p)} & \vdots \\ x_{n_1,1}^{(p)} & \cdots & x_{n_1,M}^{(p)} \end{pmatrix}. \quad (5.2)$$

La matrice  $X$  représente l'information codée suivant l'équation (3.2). Soit  $Y \in \{0;1\}^{n_2, M}$  définie comme  $x_{i,j}^{(k)}$  avec  $k = 1$  puisque la seconde couche possède un unique CbNN. La matrice d'association  $Z$  est définie telle que

$$z_{i,j} = \sum_{m=1}^M X_{i,m} Y_{m,j}^t, \quad (5.3)$$

ou matriciellement  $Z = XY^t$  avec  $t$  correspondant à la transposée. Afin d'obtenir une matrice de passage binaire, les valeurs sont seuillées telles que  $\forall z_{i,j} > 0 \Rightarrow z_{i,j} = 1$ .

Le seuillage des entrées amène un biais dans la récupération à travers le produit matriciel de l'entrée et de la matrice d'association. Cependant le réseau BCAM atténue ce biais à l'aide de la correction réalisée par les réseaux CbNN en sortie de la matrice d'association. Autrement dit, la capacité correctrice du BCAM s'appuie sur la combinaison d'une mémoire associative bidirectionnelle et de réseaux CbNN.

### 5.2.1 Phase d'apprentissage

L'apprentissage d'images, ou d'autres types de données, s'effectue par décomposition de celles-ci en patches. Chacun de ces patches est codé sous forme de cliques, nommées micro-cliques par un sous-réseau CbNN associé suivant les mécaniques d'apprentissage classiques présentées dans la section (3.2.2). En outre, l'apprentissage des sous-réseaux est indépendant, induisant une parallélisation possible du processus dans la première couche. En ce qui concerne la seconde couche, une première proposition consiste à générer aléatoirement les messages, ou macro-cliques, suivant une distribution uniforme afin d'obtenir les meilleurs performances en termes de récupération dans cette couche. En effet, dans le cas où la première couche code des images, la distribution des activations est non-uniformes entraînant l'apparition d'ambiguïtés que nous réglons avec l'uniformisation des activations en seconde couche.

La méthode s'appuie sur les résultats du CbNN restreint (RCbNN) [35] qui s'inspire des mécaniques des machines de Boltzmann restreintes. En effet, le RCbNN est constitué de deux couches dont seuls les neurones de couches différentes sont connectés. Les cliques sont inter-couches et non intra-couches. L'activation des neurones de la seconde couche est aléatoire suivant une distribution uniforme maximisant les distances de Hamming entre mots de codes générés. La phase de récupération consiste à itérer le passage d'une couche à une autre en suivant les mécaniques propres au CbNN. Autrement dit, l'utilisation de la matrice de passage entre les couches consiste à compléter l'inter-clique à l'aide des neurones des deux couches qui la composent.

Le modèle BCAM peut être considéré comme une machine de Boltzmann non restreinte. En effet, les neurones d'une même couche sont connectés entre eux et la matrice de passage  $Z$  correspond aux connexions des neurones entre deux couches. En comparaison au RCbNN, le BCAM ajoute une phase de correction supplémentaire dans chaque couche. En effet, la première couche recherche les micro-cliques qui sont par la suite transférées à travers  $Z$  afin de récupérer une macro-clique et inversement comme illustré par la figure 5.2. Quand le RCbNN récupère l'inter-clique, le BCAM retrouve les intra-cliques.

L'apprentissage s'appuyant sur l'activation aléatoire des neurones en  $L_2$  implique la construction de la matrice de passage  $Z$  après la génération des messages de la seconde couche, et ce en suivant l'équation (5.3). La phase d'apprentissage est illustrée par la figure 5.1, et l'algorithme 3 détaille les étapes.

La seconde proposition consiste à activer les neurones de la seconde couche à l'aide d'une matrice générée aléatoirement agissant comme une projection aléatoire. Cette proposition est étudiée dans la prochaine section.

---

**Algorithme 3** BCAM : Phase d'apprentissage
 

---

- 1:  $Subnet(k)^{(1)} \leftarrow x^{(k)}$  : Apprentissage des patches encodés en micro-cliques par chaque sous-réseau CbNN de  $L_1$ .
  - 2: Sélection aléatoire des neurones à activer pour  $L_2$ .
  - 3:  $net^{(2)} \leftarrow Y$  : Apprentissage des macro-cliques par  $L_2$ .
  - 4:  $Z = XY^t$  : Construction de la matrice d'association entre les couches.
- 

### 5.2.2 Phase de récupération

La récupération des messages dans la nouvelle mémoire associative est effectuée par itérations de passages entre les couches  $L_1$  et  $L_2$ , puis inversement jusqu'à convergence (lorsque les neurones activés sont les mêmes entre deux itérations successives).

Soit  $I^{(L)}$  une information en entrée de la couche  $L$ , on pose  $\hat{I}^{(L)}$  la sortie après correction par cette couche. Le passage bidirectionnel d'une couche à l'autre est réalisé par produit avec

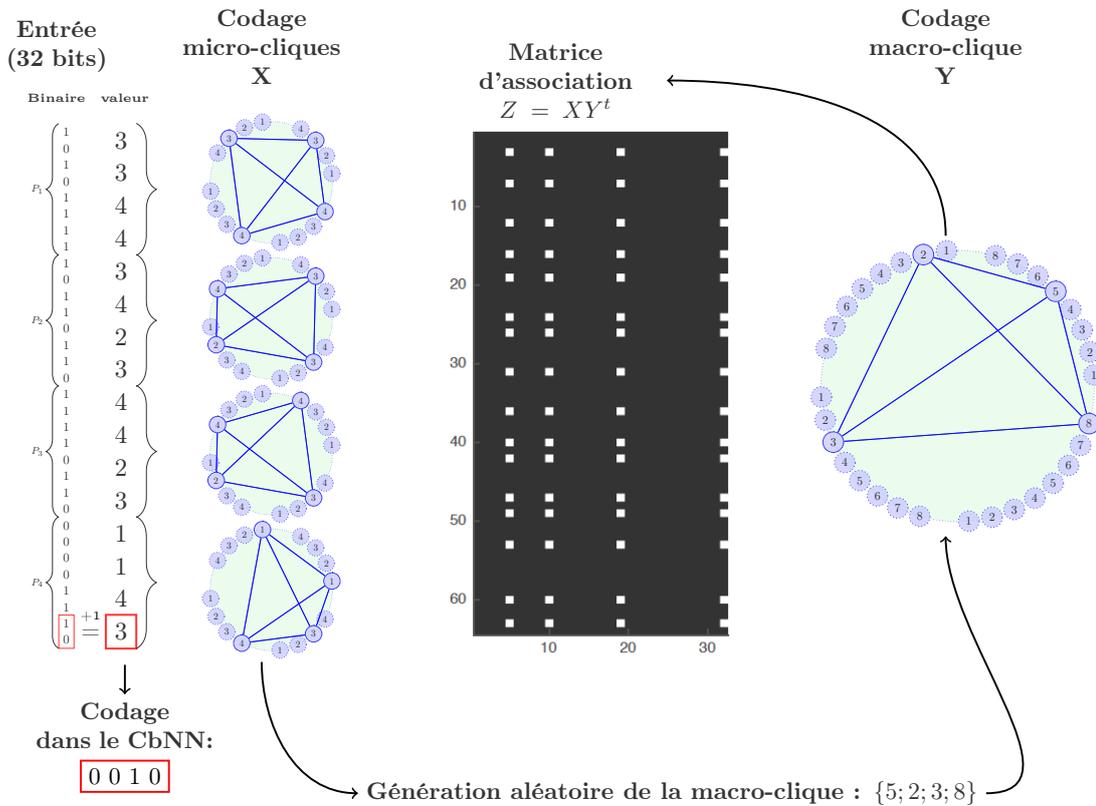


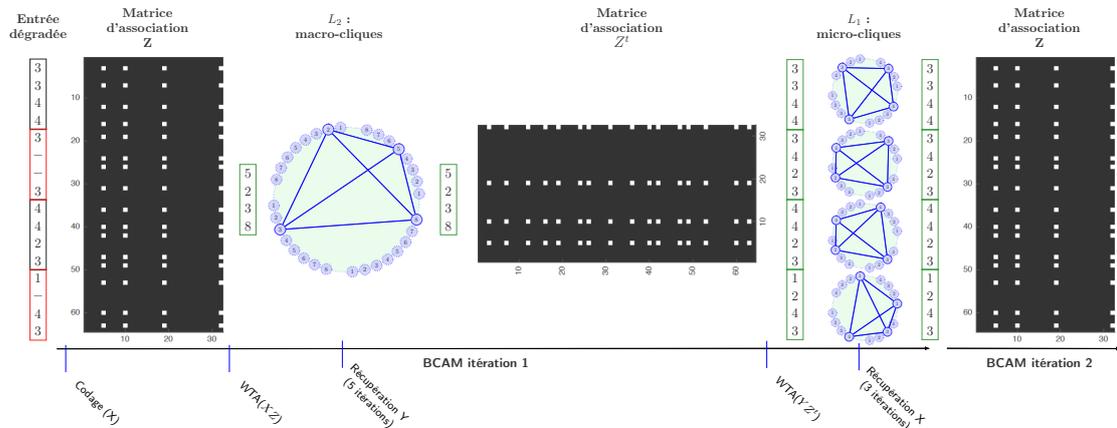
FIGURE 5.1 – Déroulement de la phase d'apprentissage du BCAM pour un message donné.

la matrice d'association  $Z$ ,

$$I^{(2)} = \hat{I}^{(1)} Z \iff I^{(1)} = \hat{I}^{(2)} Z^t \quad (5.4)$$

Ce produit est équivalent à l'application d'une règle dynamique *Sum-of-Sum* (SOS, équation 3.9). Afin que le message transformé par les poids soit interprétable par le CbNN d'une couche, nous effectuons un filtrage. Celui-ci est réalisé à l'aide d'une règle d'activation WTA (équation 3.11), ou GWTA (équation 3.12 selon l'architecture choisie) sur les scores obtenus à travers le produit matriciel de l'équation (5.4). Autrement dit, le passage d'une couche à une autre permet de réaliser une première correction suivant les mécaniques du CbNN. La figure 5.2 illustre la phase de récupération pour un message appris partiellement effacé, et l'algorithme 4 détaille les étapes.

L'usage de cette matrice d'association  $Z$  apporte au réseau BCAM une capacité de généralisation et lui permet de récupérer une image apprise à partir d'une entrée quelconque. En effet, si l'image est connue et dégradée, le réseau récupère l'image apprise au préalable. Si l'image est inconnue, dégradée ou non, le réseau BCAM est capable de converger vers une image de sa mémoire, contrairement au CbNN initial. Rappelons, le problème du CbNN initial lorsqu'il est confronté à une image non apprise soumise au réseau : dans ce cas, la



**FIGURE 5.2 –** Déroulement de la phase de récupération du BCAM pour un message appris partiellement effacé.

probabilité de produire un chevauchement de clique dépend du nombre de neurones communs à l'image qui appartiennent pourtant à des cliques différentes. C'est le manque d'a priori global que nous corrigeons ici à l'aide de la seconde couche du réseau BCAM. Ainsi, la macro-clique code l'association des micro-cliques en une seule et même clique afin d'éviter le chevauchement de cliques.

---

#### Algorithme 4 BCAM : Phase de récupération

---

- 1: **Pour** iter = 1 :maxIterations **faire**
  - 2:     Passage par la matrice d'association Z.
  - 3:     Filtrage (équivalent règle d'activation)
  - 4:     Récupération par  $L_2$ .
  - 5:     Retour à  $L_1$  via la transposée de Z.
  - 6:     Récupération des données par  $L_1$ .
  - 7: **fin Pour**
- 

Le défaut du réseau BCAM est sa génération aléatoire des macro-cliques entraînant une disjonction dans le cas de données corrélées. En effet, avec l'exemple des images de chiffres manuscrits MNIST, deux images de classe identique seront codées en des cliques entièrement différentes dans la seconde couche du BCAM à cause de cette génération aléatoire. De la même manière, deux images de classes différentes peuvent obtenir des cliques similaires du fait de la distribution aléatoire des activations neuronales de la seconde couche. Par conséquent, la problématique de l'apprentissage est de justifier l'activation d'un neurone par rapport à un autre dans la seconde couche du modèle. Cette problématique d'apprentissage fait apparaître la problématique de récupération liée à la transformation de dimension de l'information passant d'une couche à une autre. En effet, le passage d'une dimension à une autre est réalisé brutalement, car un neurone de la seconde couche doit représenter l'en-

semble des neurones actifs dans un sous-réseau. La réduction de dimension induit un biais durant la phase de récupération. Par conséquent, nous présentons dans la prochaine section une évolution du modèle BCAM afin de résoudre les problématiques décrites.

## 5.3 Évolution du modèle

L'évolution du réseau BCAM se traduit par l'utilisation d'une matrice aléatoire afin d'extraire des caractéristiques à travers une réduction de dimension et une méthode de récupération basée sur cette même matrice. La solution s'appuie sur la méthode de l'acquisition comprimée (ou *Compressive Sensing*, CS) qui permet, à l'aide d'une matrice de passage  $\phi$ , de transformer un vecteur  $x$  en un vecteur  $y$  de dimension réduite. Une optimisation est réalisée afin de récupérer le vecteur  $x$  à partir de  $y$  à travers la matrice  $\phi$ . La récupération est à haute probabilité sans perte si et seulement si le vecteur  $x$  est parcimonieux [27], ce qui est le cas des micro-cliques.

La théorie du CS [39] [128] s'appuie sur deux principes que sont la parcimonie et l'incohérence. Le premier exprime l'idée que les données peuvent être plus petites que suggérées par la bande passante par lesquelles elles sont transmises. Par conséquent, ces données peuvent être représentées de manière plus concise si elles sont projetées dans une base appropriée. Ce principe de parcimonie rejoint l'idée de codage parcimonieux défini par Olshausen [106]. Le second principe, l'incohérence, étend la dualité entre deux espaces, par exemple le temps et la fréquence. Le principe exprime l'idée que des signaux avec une représentation parcimonieuse doivent être projetés dans le domaine dans lequel ils sont acquis afin d'être plus denses, comme par exemple un Dirac du domaine temporel projeté sous forme de fréquence [25] [26]. Par conséquent, en faisant l'hypothèse qu'un signal est parcimonieux (premier principe), ce dernier peut être échantillonné à travers une matrice générée aléatoirement (second principe). En effet, une matrice aléatoire est incohérente avec presque tous les types de données dont les images. La matrice maximise donc l'incohérence entre l'image soumise et sa base de représentation, telle que la transformée en ondelettes d'images pour réaliser une compression de ces dernières.

### 5.3.1 Méthode d'acquisition comprimée

L'intégration dans le réseau BCAM de la méthode CS permet de justifier l'activation d'un neurone dans la couche supérieure suivant une base commune pour tous les exemples, contrairement au modèle initial appelé BCAM-BAM dont les macro-cliques sont toutes indépendantes et identiquement distribuées. Le passage inverse dans la première couche est effectué par minimisation itérative et non par l'utilisation d'une matrice d'association. Soit  $\phi$  la matrice générée aléatoirement, et  $X$  le signal d'entrée (les micro-cliques), le signal  $Y$  (une

macro-clique) est obtenu dans la nouvelle base selon l'équation

$$Y = X\phi. \quad (5.5)$$

La minimisation permettant à partir de  $X$  de récupérer  $x$  résulte de l'application de la norme  $l_1$  sur l'équation

$$\min \|X\|_{l_1} \text{ tq } \|Y - \phi X\|_2 \leq \epsilon. \quad (5.6)$$

La norme  $l_1$  est une contrainte permettant de reconstruire le signal  $X$  parcimonieusement. A l'instar de la haute probabilité à récupérer sans perte le signal initial, la correction des données par les sous-réseaux de la première couche du BCAM reste négligeable. En outre, si la correction en seconde couche n'est pas réussie, l'optimisation à travers la méthode CS entraîne la reconstruction à l'entrée de la première couche d'un signal inconnu et incohérent pour les sous-réseaux. Par conséquent, il est nécessaire qu'une clique ait été récupérée en  $L_2$  sous peine que la correction complète du réseau BCAM échoue. Dans le cas de la méthode CS, un maximum de deux itérations est nécessaire (sous la condition d'une réussite de la récupération en  $L_2$ ).

L'usage du CS au sein du BCAM nécessite diverses modifications au réseau initial. En effet, la proposition originelle du CS concerne une matrice aléatoire  $\phi$  composée de réels. Ainsi, le passage d'un vecteur  $X \in \{0,1\}$  entraîne la création d'un vecteur  $Y \notin \{0,1\}$  et par conséquent non interprétable par un CbNN sans passer par une étape supplémentaire de quantification. En outre, une quantification du vecteur  $Y$  produirait un biais trop important pour que l'optimisation de récupération du vecteur  $X$  soit effective.

Il est impossible de s'appuyer sur des méthodes de *compressive sensing* binaire [71] [96] dont seule la matrice  $\phi$  est binaire. En effet, les méthodes reposent sur des vecteurs  $X$  et  $Y$  à valeurs réelles. Par conséquent, ces méthodes nécessitent une quantification supplémentaire dans la seconde couche du réseau, ce qui n'est pas envisageable. Notre intérêt se focalise donc sur le CS qui s'appuie sur des mesures de 1 bits. En effet, il est possible de récupérer un signal binaire  $X \in \mathbb{B}$  de dimension  $N$  et de parcimonie d'ordre  $K$  à partir de mesures quantifiées  $Y \in \{-1;1\}^M$  (1 bit) à l'aide d'une matrice  $\phi^{N \times M} \sim \mathcal{N}(0,1)$ . La matrice  $\phi$  est donc non binaire. Sa binarité à travers le seuillage des entrées apporte un biais qui peut être corrigé à travers l'accentuation de la parcimonie du vecteur  $X$ .

Dans le but de récupérer le vecteur d'entrée  $X$  à partir de  $Y$  à l'aide de la matrice  $\phi$ , une étape de minimisation est nécessaire. La fonction de coût associée est l'équation (5.6). Compte tenu de notre besoin de réduire les ressources de calcul nécessaires au minimum, ce type d'optimisation n'est pas souhaité. Cependant, la récupération du vecteur initial  $X$  à partir de  $Y$  peut être effectuée par itération à l'aide de l'algorithme de seuillage brutal et itératif (ou *Iterative Hard Thresholding*, IHT) [18]. Cet algorithme s'appuie sur la minimisation par moindres carrées entre la sortie connue et la sortie souhaitée selon la matrice de passage, sous contrainte que le vecteur d'entrée soit parcimonieux. Ainsi, pour une itération  $i$ , la première étape correspond à calculer l'erreur quadratique des vecteurs telle que  $\|Y - \phi X\|_2^2 / 2$ ,

à l'aide d'une descente de gradient,  $a^{i+1} = X^i + \phi^t (Y - \phi X)$ . La seconde étape impose la parcimonie en projetant  $a^{i+1}$  sur la sphère  $L_0$  afin de ne sélectionner que les  $K$  entrées possédant le plus d'amplitude. Par conséquent, le IHT pour l'acquisition comprimée revient à résoudre l'équation de minimisation

$$\arg \min_u \frac{1}{2} \|Y - \phi u\|_2^2 \text{ tq } \|u\|_0 = K. \quad (5.7)$$

La méthode de CS basée sur des mesures de 1 bits repose sur une modification de l'algorithme IHT, c'est le IHT binaire (ou *Binary Iterative Hard Thresholding*, BIHT) [72]. Le BIHT modifie la première étape de l'algorithme IHT afin d'imposer la cohérence du signal reconstruit en respectant la contrainte binaire. L'algorithme est initialisé en posant  $X^0 = 0$  et définit  $\hat{Y}$  le signal de sortie binaire qui est seuillé par la fonction signe telle que  $\hat{Y} = \text{sign}(\phi X)$ . A l'itération  $i$ , le BIHT se déroule comme suit,

$$a^{i+1} = X^i + \frac{\tau}{2} \phi^t (\hat{Y} - \text{sign}(\phi X^i)), \quad (5.8)$$

$$X^{i+1} = \eta_K(a^{i+1}), \quad (5.9)$$

avec  $\tau$  un scalaire qui contrôle la taille du pas de la descente de gradient. La fonction  $\eta_K(a)$  calcule l'approximation des  $K$  meilleures valeurs de  $a$  à travers un seuillage, ou autrement dit l'algorithme sélectionne les  $K$  plus grandes valeurs une fois le vecteur trié par ordre décroissant. Le critère d'arrêt du BIHT correspond à l'obtention de la consistance du vecteur d'entrée (lorsque la parcimonie du vecteur est atteinte), ou après un maximum d'itérations.

Dans notre cas, afin de limiter le temps de calcul pour les tests, nous choisissons un nombre maximum d'itération assez bas (égal à 15). En effet, si la cohérence du vecteur n'est pas atteinte, elle peut l'être à travers la correction des sous-réseaux de la première couche du BCAM. Cependant, afin de retrouver un signal  $X \in \{0,1\}$  à partir de mesure bipolaires  $Y \in \{-1;1\}$ , le facteur de parcimonie  $K$  du signal initial doit être connu. Le modèle BCAM permet un apprentissage parcimonieux, où le nombre de clusters nécessaires pour représenter l'information est inférieur au nombre total de clusters disponibles dans le réseau. A travers l'utilisation d'une architecture parcimonieuse, deux messages peuvent avoir un ordre de parcimonie différent. Toutefois, l'architecture parcimonieuse du BCAM n'est pas applicable à tous les jeux de données. Le modèle peut aussi être utilisé sous forme pleine où tous les clusters du réseau sont utilisés afin de représenter l'information. Sous cette forme, l'indice de parcimonie  $K$  reste le même pour tous les messages. Cependant, le décuplement de  $K$  implique une augmentation de la dimension de la matrice  $\phi$  et donc un temps de récupération plus élevé.

Une solution consiste à fixer un nombre de clusters actifs pour chaque sous-réseau. Une fois l'image quantifiée puis découpée en patchs, l'apprentissage des sous-réseaux consiste à activer uniquement les neurones des clusters actifs, soit les  $c$  valeurs les plus élevées. La

valeur de parcimonie  $K$  est primordiale en CS. En effet, le bon fonctionnement de l'optimisation nécessite pour un signal  $X \in \mathbb{B}^N$ ,  $K = \sum_i^N x_i$ , que la dimension  $M$  du vecteur  $Y$  satisfasse la propriété d'isométrie restreinte (ou *restricted isometry property*, RIP) [24] définie par l'équation

$$M \geq C_\delta K \log(N/K). \quad (5.10)$$

L'utilisation de la méthode CS au sein du BCAM entraîne une modification de l'architecture du réseau. En effet, tous les paramètres du réseau découlent d'un sous-ensemble de paramètres que sont la résolution des images à apprendre et la taille de leurs patches. Autrement dit, contrairement à d'autres réseaux de neurones, la topologie du réseau BCAM n'est pas empirique.

### 5.3.2 Topologie adaptative

Soit  $I$  une information de dimension  $n \times m$ . Le nombre de patches dépend de la taille des blocs en ligne  $b_r$  et colonne  $b_c$  que nous souhaitons, ainsi que du pas  $t$  entre ceux-ci, soit

$$p = \left( \frac{n - b_r}{t} + 1 \right) \left( \frac{m - b_c}{t} + 1 \right). \quad (5.11)$$

Les caractéristiques de la couche  $L_1$  dépendent de ces paramètres fixés par l'utilisateur. Le nombre des sous-réseaux est égal à  $p$  et le nombre total de clusters dans un sous-réseau est  $\chi_1 = b_r \times b_c$  avec  $c_1$  les clusters actifs pour exprimer l'information. La quantité de neurones par cluster est une constante définie à travers la quantification désirée,  $l_1 = q$ . La dimension d'une matrice d'adjacence d'un sous-réseau est  $\dim(A_1) = (\chi_1 l_1)^2$ . En ce qui concerne l'étape de CS, l'entrée et la sortie de l'ensemble des sous-réseaux (les micro-cliques,  $X$ ) sont de dimension  $N$ , et nous posons  $K$  la somme des neurones actifs pour représenter l'information. Si l'architecture est pleine, le facteur de parcimonie est  $K_f = \chi_1 p$ , sinon,  $K_s = c_1 p$ . La dimension  $M$  du vecteur  $Y$  dépend de la contrainte RIP définie par l'équation (5.10). Les caractéristiques de la seconde couche  $L_2$  dépendent de ces dernières variables, avec  $\chi_2 = M/s$  et  $s$  la taille de la séquence binaire de  $Y$  allouée à un cluster. Le nombre de neurones par cluster est  $l_2 = 2^s$ . A l'instar des sous-réseaux de la première couche, la dimension de la matrice d'adjacence en  $L_2$  est de  $\dim(A_2) = (\chi_2 l_2)^2$ .

Par exemple, dans le cas d'une application de reconstruction d'une image MNIST, le modèle parcimonieux est à privilégier. En effet, le codage du fond des images correspond à un gaspillage de mémoire. Par conséquent, les paramètres de l'architecture du réseau dépendent du jeu de données à apprendre. La résolution des images de MNIST est de  $28 \times 28$  pixels de profondeur 8 bits. Nous posons  $p$  le nombre de patches souhaités pour une image  $I$  donnée. Soit  $p = 4$ , ce qui implique une décomposition de l'image en quatre patches de résolution  $14 \times 14 = 196$ . Par conséquent, la première couche du modèle BCAM est composée

de  $p = 4$  sous-réseaux, chacun constitué de  $\chi_1 = 196$  clusters, et leur nombre de neurones dépend de la quantification souhaitée (appliquée à l'aide des K-moyennes). Soit  $l_1 = 16$ , pour une profondeur de 4 bits au lieu de 8 bits initialement. Le nombre de clusters actifs  $c_1$  représentant l'information dépend de l'architecture. Soit l'architecture est pleine et  $c_1 = \chi_1$ , soit elle est parcimonieuse et on pose  $c_1$  comme la moyenne des neurones nécessaires pour représenter l'information effective des images, en l'occurrence  $c_1 = 40$  dans notre exemple. En ce qui concerne l'acquisition comprimée (CS), la dimension  $N$  du vecteur  $X$  d'entrée est  $N = \chi \times l \times p = 12\,544$  et l'ordre de parcimonie  $K$  est égal au nombre de clusters nécessaires pour décrire l'information des différent patches,  $K = \sum_i^N x_i = c_1 p \leq \chi p$ . À l'aide de la contrainte RIP (équation (5.10)), la dimension  $M$  est estimée à  $M \approx 4\,000$  avec  $C_\delta = 5$ . L'architecture de la seconde couche dépend de la dimension  $M$  du vecteur  $Y$  résultant du produit de l'entrée  $X$  et de la matrice  $\phi$ , en l'occurrence 4 000. La séquence bipolaire  $Y \in \{-1; 1\}$ , obtenue à travers l'utilisation de la fonction signe, est ensuite transformée en séquence binaire  $Y \in \{0; 1\}$  et finalement découpée en sous-séquences de taille  $s$ .

Le paramètre  $s$  gère la capacité de la mémoire du réseau en  $L_2$ . En effet, qu'il soit fixé à  $s = 8$  ou  $s = 5$ , cela conduit respectivement à 500 clusters contenant 256 neurones chacun avec une matrice d'adjacence associée de dimension  $128\,000^2$ , ou 800 clusters comportant 32 neurones entraînant la dimension de la matrice d'adjacence à  $25\,600^2$ . Par conséquent, le paramètre  $s$  est important puisque il gère la dimension de la matrice d'adjacence de la seconde couche. Afin d'améliorer la gestion de la mémoire, il est donc nécessaire de minimiser  $s$  pour que la dimension de la matrice d'adjacence soit raisonnable. La borne inférieure de  $s$  est égale au nombre de neurones dans un cluster représentant les deux valeurs possibles de la binarité soit, "0" et "1", et donc  $s$  tel que  $l_2 = 2$  soit  $s = 1$ . Pour trouver la borne supérieure en fonction d'une taille de matrice d'adjacence raisonnable, on pose  $C$  cette variable, le problème se pose comme

$$\sqrt{\dim(A_2)} \leq C. \quad (5.12)$$

En remplaçant les variables par le paramètre que l'on souhaite minimiser,

$$\chi_2 l_2 \leq C \Leftrightarrow \frac{M}{s} 2^s \leq C. \quad (5.13)$$

Nous cherchons donc à résoudre l'inéquation

$$\frac{2^s}{s} \leq \frac{C}{M}, \quad (5.14)$$

dont la solution est

$$s \leq -\frac{W_{-1}\left(\frac{\log(2)}{C/M}\right)}{\log(2)}, \quad (5.15)$$

avec  $W_{-1}(x)$  la branche inférieure de la fonction  $W$  de Lambert, appelée fonction omega ou produit logarithmique. La fonction de Lambert correspond à la relation inverse de la

fonction  $f(W) = Wexp(W)$ . Afin d'obtenir une large diversité dans la mémoire de la couche  $L_2$ , nous proposons de prendre la partie entière de la borne supérieure de  $s$ .

Une fois le paramètre  $s$  déterminé, le nombre de clusters est égal au nombre de sous-séquences  $\chi_2 = M/s$ . Le nombre de neurones par cluster dans la couche  $L_2$  est donc  $l_2 = 2^s$ . Le réseau de cette couche est plein,  $c_2 = \chi_2$ . En effet, le choix d'architecture s'explique par la faible probabilité de rencontrer une séquence de taille  $s$  possédant une somme égale à 0, et ce d'autant plus quand il est souhaité que  $s$  soit élevé (sous contrainte de  $C$ ) pour maximiser la diversité du réseau. La figure 5.3 illustre la phase d'apprentissage du BCAM avec la méthode CS, et l'algorithme 5 détaille les étapes.

---

**Algorithme 5** BCAM-CS : Phase d'apprentissage
 

---

- 1:  $Subnet(k)^{(1)} \leftarrow x^{(k)}$  : Apprentissage des patchs codés en micro-cliques par chaque sous-réseau CbNN de  $L_1$ .
  - 2:  $K = \sum_k c^k$  : Parcimonie
  - 3:  $M \geq C_\delta K \log(N/K)$  : Définition de la taille de la matrice de passage à travers la parcimonie  $K$  des données de  $L_1$  et sa dimension totale  $N$ , avec  $C_\delta = 5,5$  une constante.
  - 4:  $\phi \sim \mathcal{N}(0,1)$
  - 5:  $Y = \text{sign}(X\phi)$  tel que  $\forall y \in \{0,1\}$  : Séquence binaire  $Y$  issue du passage de  $X$  via  $\phi$ .
  - 6:  $s \leq -W_{-1} \left( \frac{\log(2)}{C/M} \right) / \log(2)$  : Définition du découpage de la séquence  $Y$  pour le codage  $L_2$ .
  - 7: Transformation décimale des séquences binaires de  $Y$  découpées par  $s$  afin de sélectionner les neurones à activer pour  $L_2$ .
  - 8:  $net^{(2)} \leftarrow Y$  : Encodage puis apprentissage des macro-cliques par  $L_2$ .
- 

Le tableau 5.2 détaille les architectures possibles du BCAM avec la méthode CS pour l'exemple d'apprentissage d'images MNIST.

Type	L1			CS				L2	
	$\chi_1$	$c_1$	$l_1$	$N$	$K$	$M^{(5.10)}$	$s^{(5.15)}$	$\chi_2$	$l_2$
Base	196	196	16	12 544	784	11 000	5	2 200	32
Plein	196	196	16	12 544	784	11 000	5	2 200	32
Parcimonieux	196	40	16	12 544	120	4 000	5	800	32

**TABEAU 5.2** – Exemples d'architectures avec l'utilisation du Compressive Sensing au sein du BCAM pour  $p = 4$  patchs de résolution  $b_r = 14, b_c = 14$  et une taille de pas de  $t = 14$ .

En ce qui concerne la phase de récupération, l'algorithme 6 détaille les étapes, et la figure 5.4 illustre le mécanisme de récupération. L'usage de la méthode CS au sein du BCAM amène une topologie de son architecture qui s'adapte directement aux données.



**Algorithme 6** BCAM-CS : Phase de récupération

- 
- 1: **Pour** iter = 1 :maxIterations **faire**
  - 2:     Passage par la matrice d'association  $\phi$ .
  - 3:     Filtrage (équivalent règle d'activation)
  - 4:     Récupération par  $L_2$ .
  - 5:     Retour à  $L_1$  via  $\phi$  avec BIHT.
  - 6:     Récupération des données par  $L_1$ .
  - 7: **fin Pour**
- 

### 5.3.3 Variantes du modèle BCAM

Les deux modèles principaux sont le BCAM-BAM et le BCAM-CS. Le premier modèle repose sur l'utilisation d'une matrice d'association bidirectionnel (BAM), et le second sur l'acquisition comprimée (CS). Nous proposons dans cette section des variations de ces modèles. En effet, le BCAM-BAM est limité en termes de diversité du fait de la contrainte sur le nombre de clusters en seconde couche. Pour le BCAM-CS, la matrice de passage  $\phi$  de dimension élevée et à valeurs réelles requiert une consommation élevée de mémoire.

#### 5.3.3.1 BCAM-BAM

Le réseau BCAM-BAM utilise une matrice binaire faisant transiter l'information bidirectionnellement entre les couches à l'aide d'une matrice d'association  $Z$ . Cependant, le nombre de clusters  $\chi_2$  dans la seconde couche est égal au nombre de patches  $p$  après décomposition des données. Pour cela, nous appellerons désormais ce modèle BCAM-BAM $p$ . A l'instar de la génération aléatoire des macro-cliques, une problématique du BCAM-BAM $p$  est l'architecture pleine de la seconde couche. Bien que le nombre de neurones  $l_2$  puisse être calculé afin de satisfaire une densité équivalente à la première couche, la diversité est limitée. Nous proposons une variante telle que le nombre de clusters en seconde couche  $\chi_2$  puisse être différent du nombre de patches  $p$ . Ainsi, la variante est nommée BCAM-BAM $\infty$ . La relaxation de cette contrainte permet au modèle BCAM-BAM $\infty$  de posséder une architecture parcimonieuse pour la seconde couche afin de maximiser sa diversité.

#### 5.3.3.2 BCAM-CS

En ce qui concerne le modèle BCAM-CS, de par sa matrice de passage  $\phi$  de haute dimension à valeur réelle, mais aussi l'architecture pleine de sa seconde couche, le coût mémoire est supérieur au modèle BCAM-BAM comme en atteste la figure 5.5b. De ce fait, ce modèle est baptisé BCAM-CSfull.

Un moyen pour réduire le coût mémoire est de diminuer les dimensions de la matrice  $\phi$

afin de l'appliquer à l'ensemble des micro-cliques non pas toutes ensemble, mais par sous-réseau. Par conséquent, la taille de  $\phi$  est réduite par rapport à la dimension des données d'entrée en fonction du nombre de sections dans ces dernières. La dimension du vecteur de sortie  $Y$  reste inchangée. La construction de  $Y$  est réalisée par parties, chacune issue du passage d'une micro-clique par la matrice  $\phi$  réduite. Soit la micro-clique  $x^{(k)}$  du sous-réseau  $k$ , sa dimension est  $n = N/p$ , et son coefficient de parcimonie est  $c^k = K/p$ . La matrice  $\phi^k$  est donc réduite proportionnellement au nombre de sections  $p$ . La dimension  $m$  de  $\phi^k$  est réduite par  $p$  à travers la contrainte RIP qui dépend de  $n$  et  $c_1^{(k)}$  eux-mêmes réduits par  $p$ , soit  $m = M/p$ . Chaque sous-réseau est associé à une matrice  $\phi^{(k)}$  réduite.

Une première variante du modèle à base de CS consiste à considérer une unique matrice  $\phi$  réduite appliquée à l'ensemble des sous-réseaux, le modèle se nomme alors BCAM-CSmini. Une seconde variante est d'appliquer  $p$  matrices  $\phi$  distinctes afin d'extraire des caractéristiques différentes pour chaque sous-réseau, c'est le BCAM-CSminiS.

Les architectures des différents modèles utilisés sont détaillées dans le tableau 5.5a. Ce dernier contribue à préciser la dimension des matrices de passage  $Z$  ou  $\phi$  et l'architecture de la seconde couche du réseau en fonction du modèle. Les paramètres sont adaptés en fonction du type de passage utilisé, *c.-à-d.* BAM ou CS. En effet, pour la méthode CS, la dimension de la matrice  $\phi$  est différente selon que son application porte sur l'ensemble des sous-réseaux ou par parties. Pour la génération des paramètres d'architectures de la seconde couche du réseau BCAM-CS, nous contraignons son nombre de neurones total à  $C = 20000$ . Par conséquent, les réseaux avec la méthode CS par parties, de par la faible dimension de la matrice  $\phi$ , nécessitent un faible nombre de clusters, ceux-ci devant posséder un plus grand nombre de neurones pour compenser la réduction initiale. Cette différence permet aux réseaux de stocker plus de messages puisque, comme présenté dans la section (3.2.2) dédiée à la présentation du CbNN avec la figure 3.2, le nombre de neurones par cluster est le paramètre crucial dans le nombre de messages que le réseau peut apprendre.

La prochaine section présente les expérimentations menées à l'aide du réseau BCAM sur des données artificielles, puis sur divers jeux de données d'images.

## 5.4 Expérimentations

Les expérimentations se déroulent en deux étapes. La première a pour but de tester la capacité correctrice du réseau sur des données apprises au préalable. A cet égard, nous utilisons des données générées aléatoirement suivant une distribution uniforme, puis gaussienne (non-uniforme). Une dégradation sera appliquée afin de déterminer si le réseau BCAM conserve la capacité correctrice du CbNN initial. La seconde étape consiste à mesurer la généralisation de l'information réalisée par le BCAM. Afin de démontrer que le modèle BCAM est bien en mesure de généraliser l'information, il faut être capable d'analyser les

reconstructions et de définir si le résultat est satisfaisant. En effet, à défaut de classification correcte, l'image récupérée est-elle une approximation de l'entrée inconnue ? Notre mesure s'appuie sur le critère d'erreur quadratique moyenne (ou *Mean Squared Error*, MSE) calculé sur les images récupérées, en complément des résultats en termes de taux de classification. Les expérimentations sont ici menées sur des images inconnues non dégradées afin de tester la capacité de généralisation du nouveau modèle. En outre, la classification pour le BCAM est effectuée à l'aide de la méthode Softmax en suivant le diagramme de la figure 4.15. Les résultats sont comparés à une référence calculée suivant le diagramme de la figure 4.12a. Contrairement à la section relative à la classification présentée dans le chapitre précédent, le CbNN n'est ici pas considéré comme un classifieur. Le réseau à cliques n'effectue qu'une récupération d'image, c'est le Softmax qui classe les images récupérées.

Comme lors des expérimentations sur la pénalisation et la classification avec le CbNN, le jeu de données utilisé est MNIST. En effet, ce jeu de données permet de profiter d'une première couche composée de sous-réseaux possédant une architecture parcimonieuse. Avant apprentissage, les images passent par une étape de transformation. Celle-ci correspond à la quantification puis au découpage des images. Le découpage des images attribue le nombre de sous-réseaux composant la première couche du BCAM. Les  $p$  sous-réseaux possèdent une architecture identique. Le nombre de clusters  $\chi_1^{(k)}$  ( $k = 1, \dots, p$ ) est égal au nombre de pixels du patch d'image, et le nombre de neurones composant ces clusters est égal à la quantification appliquée aux images, soit  $l_1^{(k)} = q$ , par la méthode des K-moyennes.

L'architecture de  $L_2$  dépend de  $L_1$  et de la méthode utilisée. En effet, pour la méthode BAM $p$ , le nombre de clusters est égal au nombre de sous-réseaux, soit  $\chi_2 = p$  et le nombre de leurs neurones est calculé à travers l'équation (5.1) afin d'équilibrer la densité du premier réseau. Pour la méthode BAM $\infty$ , le nombre de clusters en  $L_2$  est défini arbitrairement afin d'obtenir une consommation mémoire équivalente au modèle CbNN initial. En ce qui concerne la méthode CS, le nombre de clusters en  $L_2$  dépend de la taille de la matrice de passage  $\phi$  qui elle-même dépend de la dimension des données soumises à apprentissage. Pour les neurones  $l_2$  du modèle CS, le nombre de clusters dépend du découpage de la séquence obtenue à travers le passage de la matrice  $\phi$ . Ce découpage est contraint par la création d'une matrice d'adjacence pour  $L_2$  raisonnable. L'architecture en  $L_2$  est parcimonieuse pour BAM $\infty$  et pleine pour les autres méthodes : BAM $p$  et à base de CS.

La phase de récupération utilise la règle dynamique SoS pour tous les réseaux. La règle d'activation dépend quant à elle des architectures. Par conséquent, le CbNN initial et la première couche du BCAM utilisent le GkWTA quand la seconde couche dépend de la méthode utilisée. En effet, la règle GkWTA est appliquée si le réseau est parcimonieux sinon il s'agit de la règle WTA.

A l'instar de la section précédente, les paramètres du modèle BCAM s'adaptent à un sous-ensemble de paramètres de base que sont la taille des patches et le pas entre ceux-ci.

Ensuite, d'autres variables sont à prendre en compte en fonction de la méthode utilisée pour le passage de l'information d'une couche à une autre.

Pour les expérimentations sur les données artificielles, les résultats consistent en une moyenne des corrections réussies. Si le réseau corrige sans aucune erreur l'entrée connue et dégradée, le résultat est 1, sinon 0. Les expérimentations sur les images reposent sur la moyenne des scores MSE et du taux de classification en fonction du nombre de messages appris.

Avec la méthode du CS la dimension  $N$  restera la même, et ce quel que soit le nombre de patches (dans le cas d'un chevauchement inexistant). La dimension  $N$  est égale au produit de la résolution et de la quantification. La dimension reste élevée en dépit d'une quantification, et ce sans chevauchement de patches. Par conséquent, le chevauchement est à proscrire. Les expérimentations sur les images concernent le jeu de données MNIST quantifiées de 8 bits à 4 bits. Le temps d'exécution est plus important avec la méthode CS du fait de l'algorithme BIHT qui effectue la récupération, en dépit de la réduction du nombre d'itérations.

Le nombre d'itérations maximal du réseau BCAM est fixé à 5, mais dans le cas où deux itérations successives obtiennent le même résultat, nous supposons une convergence du réseau et le processus itératif est arrêté. Le CbNN initial nécessite un nombre d'itérations maximum de 5, tandis que pour les sous-réseaux il est fixé à 3, et pour le réseau de la seconde couche il est fixé à 5. Les nombres d'itérations sont détaillés dans les figures 5.2 et 5.4 illustrant respectivement la récupération du BCAM-BAM et du BCAM-CS. Le nombre d'itérations de la seconde couche est plus élevé de par l'importance d'obtenir une clique dans cette couche afin que la récupération sur l'ensemble du réseau BCAM soit effective.

L'algorithme 7 présente le protocole d'expérimentation sur les images. En ce qui concerne les données générées artificiellement, la différence est que le taux de récupération est calculé à partir d'une distance de Hamming.

---

**Algorithme 7** Expérimentations avec le BCAM : Vue globale

---

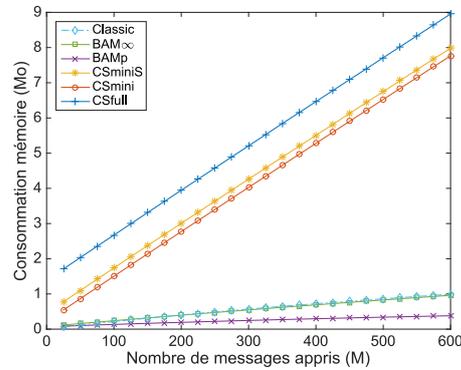
- 1: Initialisation avec la préparation des données d'apprentissage à soumettre au réseau.
  - 2:  $Data \leftarrow$  Chargement des images d'entraînement.
  - 3:  $QData \leftarrow$  Quantification.
  - 4:  $QData(1, \dots, p) \leftarrow$  Décomposition des données quantifiées en  $p$  patches. Phase d'apprentissage et récupération à travers le codage des entrées.
  - 5: Apprentissage et récupération des données à travers une méthode : BAM ou CS.
  - 6: Résultats.
  - 7: Calcul du critère MSE et classification à l'aide du Softmax.
-

### 5.4.1 Données générées artificiellement

Les données artificielles consistent en deux ensembles de taille 80 bits avec le premier comprenant une génération uniforme et le second une génération gaussienne (non-uniforme) des activations neuronales dans les clusters actifs. Le tableau 5.5a présente les architectures des différents modèles de mémoire associative utilisés avec un découpage des données en  $p = 4$  parties pour le réseau BCAM. La dimension de 80 bits des données nous fait utiliser une architecture pleine en  $L_1$  de  $\chi_1^{(k)} = c_1^{(k)} = 4$  clusters comportant chacun  $l_1^{(k)}$  neurones. Le CbNN initial est doté de  $\chi = 16$  clusters comportant chacun  $l = 16$  neurones.

Type	$Z/\phi$		$L_2$		
	$n$	$m$	$\chi_2$	$c_2$	$l_2$
CS full	512	249	83	83	8
CS mini/S	128	62	38	38	256
BAM p	512	128	4	4	32
BAM $\infty$	512	3 200	38	19	128

(a) Architectures

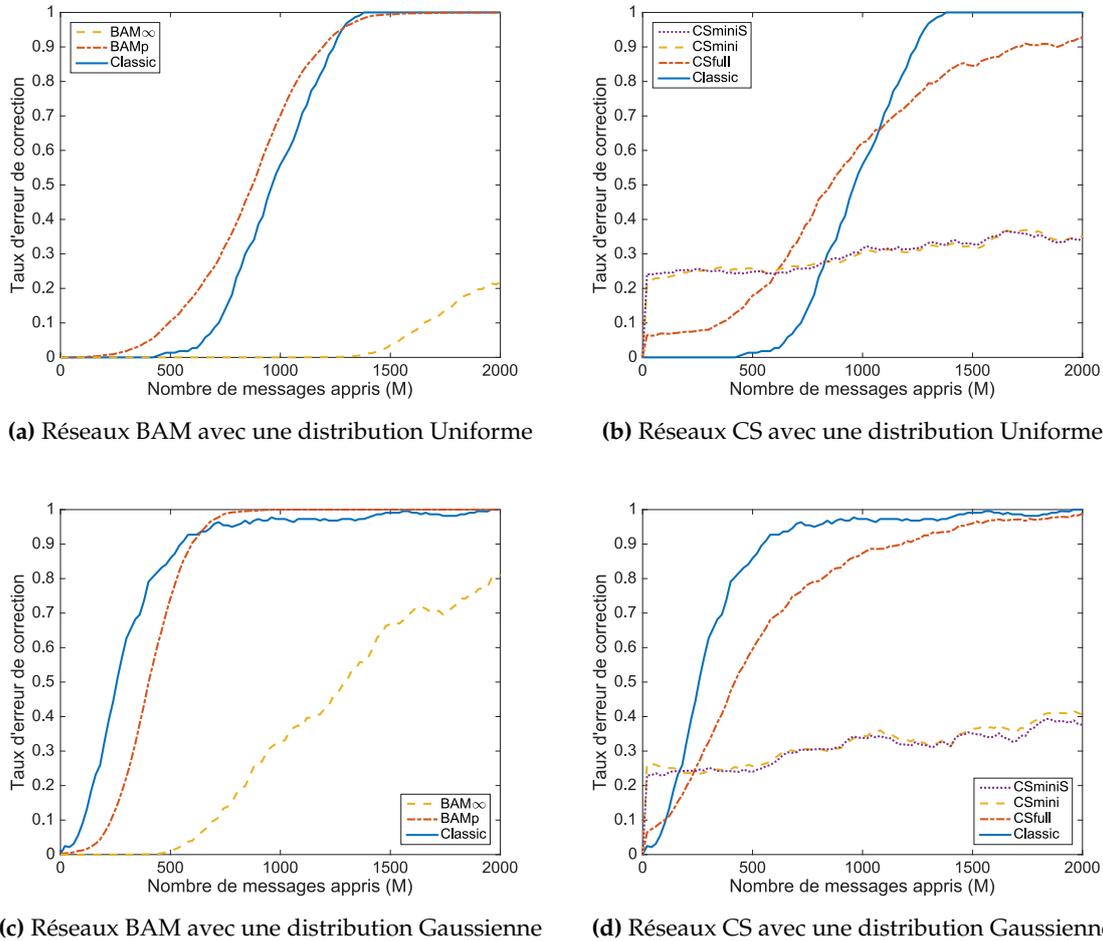


(b) Consommation de la mémoire

**FIGURE 5.5** – Architectures et consommations mémoire des réseaux BCAM utilisés pour les expérimentations sur les données uniformes et non-uniformes.

La figure 5.6 présente les résultats de récupérations sur des messages générés aléatoirement suivant deux distributions différentes, uniforme et gaussienne, en fonction des variantes possibles du réseau BCAM et des messages appris. Les résultats sont obtenus à partir de 20 séries, consistant à corriger 20 messages appris mais partiellement effacés à hauteur de 25% de l’information. La figure distingue les variantes du BCAM afin d’évaluer leurs performances de correction en comparaison au CbNN initial.

La méthode CS plein (CSfull) correspondant à une unique matrice  $\phi$  connectant l’ensemble des neurones de la première couche à ceux de la seconde obtient de meilleures performances que le CbNN initial dans le cas de l’apprentissage non-uniformes. En effet, à travers l’utilisation d’une matrice possédant des entrées non-uniformes, la méthode CS est capable de maximiser le traitement de l’information par rapport au CbNN. L’utilisation de matrices de passage associées aux sous-réseaux CbNN de la première couche du BCAM, *c.-à-d.* CSmini et CSminiS, permet de réduire la dimension de ces matrices et leurs coûts mémoire. Ces méthodes obtiennent d’excellentes performances indépendamment du remplissage du réseau et de la distribution des données. Que les matrices de passage soient les mêmes (CSmini) ou non (CSminiS) entre les sous-réseaux entraîne une différence de performances in-



**FIGURE 5.6** – Résultats de correction sur des données uniformes et non-uniformes (gaussiennes) utilisant des réseaux BCAM (20 séries avec 20 séquences de tests identiques).

signifiante, légitimant l'utilisation de la méthode CSmini pour réduire l'impact mémoire illustré à travers la figure 5.5b. A la différence des autres méthodes dont les performances de récupération sont parfaites sur l'ensemble des séries au nombre faible de messages appris, les méthodes s'appuyant sur la méthode CS par parties (CSmini/S) sont plus aléatoires. Il faut noter que la récupération à l'aide de la méthode d'acquisition comprimée s'effectue seulement à forte probabilité et par conséquent n'est pas garantie. La différence entre les méthodes CS plein et par parties provient de la diffusion d'erreurs à travers la matrice de passage  $\phi$ . En effet, la dimension supérieure de la matrice de passage pour la méthode CS plein implique une plus large diffusion. A contrario, les méthodes CS par parties réduisent les erreurs diffusées qui sont par la suite corrigées à travers les couches de CbNN du BCAM. Cependant, l'apparition d'ambiguïtés et des erreurs existantes à travers le BIHT induisent la forme linéaire des résultats de CS par parties.

La méthode  $BAM_p$  avec  $p$  le nombre de clusters dans la seconde couche du réseau BCAM en fonction des sections dans les données apprises, obtient des résultats similaires pour une consommation mémoire inférieure au CbNN initial comme illustré par la figure 5.5b. Bien que l’usage de données uniformes entraîne des performances inférieures au CbNN (Classique), nous observons l’inverse avec des données non-uniformes. La différence s’explique par l’architecture du BCAM et sa seconde couche, dont les neurones sont activés aléatoirement suivant une distribution uniforme. La seconde couche aide le réseau à faire une distinction entre les cliques non-uniformes. En ce qui concerne la méthode  $BAM_\infty$  qui possède une architecture demandant une consommation mémoire identique au CbNN initial, les résultats sont excellents. La consommation mémoire du BCAM-BAM illustrée par la figure 5.5b est identique ou moindre que celle du CbNN initial, et ce malgré la plasticité, puisque ce dernier enregistre toutes les connexions entre tous les neurones actifs en fonction des données soumises. En effet, le BCAM-BAM divise quadratiquement par  $p = 4$  ses connexions et cette différence n’est pas rattrapée en dépit de l’utilisation d’une matrice d’association et d’une nouvelle matrice d’adjacence associée à la seconde couche du réseau.

A l’instar des travaux précédents, la plasticité est primordiale. En effet, l’extension de la méthode  $BAM_p$  en  $BAM_\infty$  entraîne la possibilité de ne plus limiter le nombre de clusters en  $L_2$  par le nombre de sections des données. Les résultats illustrés à travers les figures 5.6a et 5.6c démontrent que la méthode  $BAM_\infty$  obtient les meilleures performances sur des données uniformes. Pour les données non-uniformes, le  $BAM_\infty$  obtient toujours les meilleurs résultats jusqu’à 1 000 messages, puis c’est la méthode CS par parties qui affiche les meilleures performances. En effet, la saturation de la matrice de passage du BCAM-BAM entraîne une augmentation des ambiguïtés en seconde couche, laissant ainsi le réseau incapable de retrouver une clique.

La figure 5.7 illustre les performances du BCAM-BAM $_\infty$  par rapport aux autres méthodes de la littérature. En dépit du fait que le réseau ne soit pas spécifiquement conçu pour réduire les ambiguïtés, il reste capable d’atteindre les performances des autres méthodes qui se concentrent sur la résolution de cette problématique. Le protocole d’expérimentation est identique à celui présenté dans le chapitre précédent pour comparer le modèle à couches multiples (figure 4.6). Les données sont générées artificiellement suivant une distribution gaussienne ( $\mathcal{N}(125,25)$ ) de dimension 64 bits. Dans le cas du réseau BCAM, les données sont sectionnées en 4 parties impliquant l’usage de 4 sous-réseaux CbNN dans la première couche. Les sous-réseaux comportent  $\chi_1 = 2 = c_1$  clusters composés chacun de  $l_1 = 256$  neurones. Le réseau CbNN de la seconde couche comporte  $\chi_2 = 100$  clusters dont  $c_2 = 15$  actifs pour représenter les macro-cliques. Le nombre de neurones pour ces clusters est  $l_2 = 256$ . En dehors de la méthode s’appuyant sur le codage de Huffman, le BCAM obtient les meilleures performances de récupération. De plus, le BCAM peut continuellement apprendre et ne possède pas de phases de codage et de décodage supplémentaires contrairement à la méthode de codage de Huffman. Notons que pour des performances équivalentes, voire supérieures

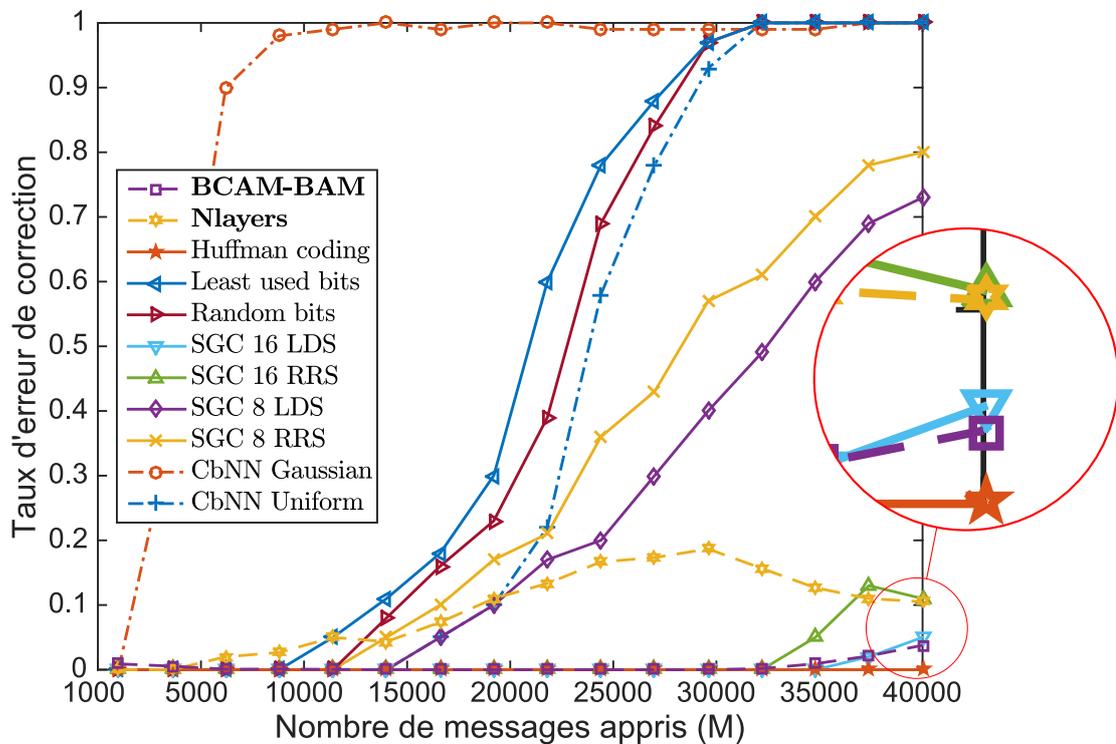


FIGURE 5.7 – Taux de récupération d'un réseau BCAM-BAM $\infty$  comparé aux méthodes de la littérature [142].

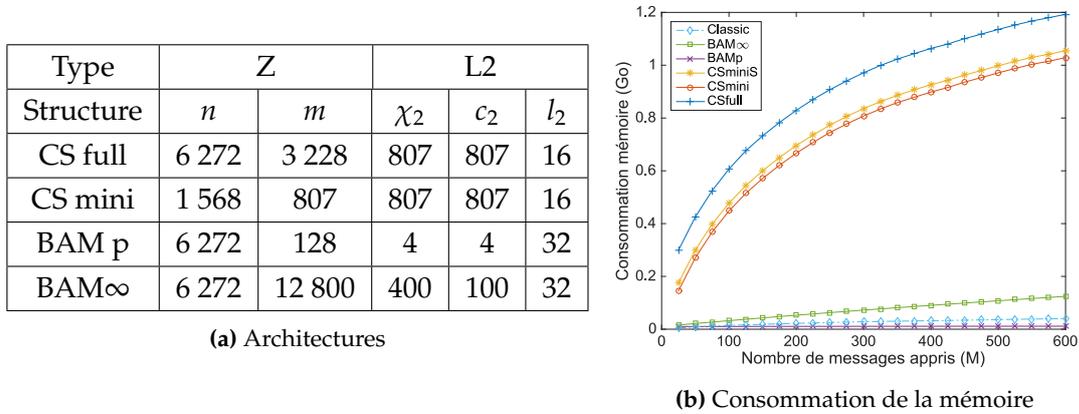
sur le long terme, le BCAM-BAM $\infty$  demande 27 648 neurones quand les réseaux à base de clones (SGC 16LDS) en nécessitent 32 768.

Au-delà des expérimentations menées sur des données artificielles pour démontrer la capacité correctrice du CbNN qui est conservée par le BCAM, nous pouvons sélectionner les méthodes viables pour le traitement de données à dimensions supérieures telles que les images. En s'appuyant sur les résultats précédents, nous limitons les expérimentations du BCAM sur les images aux méthodes CSmini et BAM $\infty$ .

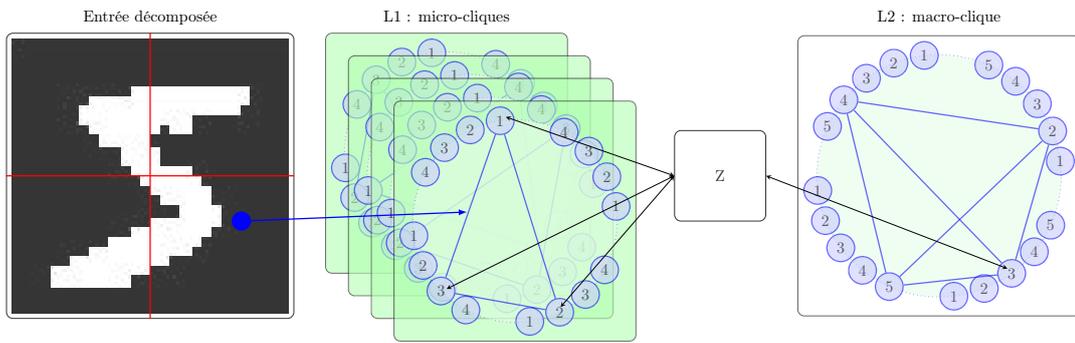
### 5.4.2 MNIST

Dans cette section nous étudions la capacité du réseau à récupérer des images inconnues à travers la classification des reconstructions du CbNN initial et du BCAM à l'aide du Soft-max entraîné au préalable sur les données quantifiées apprises par les mémoires associatives (figure 4.15).

Les expérimentations sont menées sur un réseau BCAM pourvu de quatre sous-réseaux CbNN en première couche induisant une décomposition des images MNIST en quatre



**FIGURE 5.8** – Architectures et consommations mémoire associées des réseaux BCAM utilisés pour les expérimentations sur les images MNIST.



**FIGURE 5.9** – Illustration de l'apprentissage d'une image à l'aide du réseau BCAM.

patches de dimensions égales,  $14 \times 14$ . La quantification des images réalisée à l'aide de la méthode des K-moyennes réduit la profondeur des pixels de 8 bits à 4 bits portant le nombre de neurones par clusters à  $l_1^{(k)} = 16$  pour l'ensemble des sous-réseaux avec  $k = 1 \dots p$  le nombre de patches. Le nombre de clusters est  $\chi_1^{(k)} = 14 \times 14 = 196$  avec  $c_1^{(k)} = 40$  clusters actifs. L'architecture de la première couche du BCAM est identique pour les méthodes BAM et CS. La différence provient de la dimension de la matrice de passage et l'architecture de la seconde couche. Ces dimensions sont détaillées dans le tableau 5.8a. La figure 5.8b illustre l'utilisation de la mémoire du BCAM en fonction des méthodes utilisées.

La mémoire consommée par la méthode s'appuyant sur le CS full nécessite 1,2 Go pour l'apprentissage de 600 images, ce qui justifie notre choix de ne pas l'utiliser dans les tests de récupérations d'images. Cette quantité de mémoire est due à la dimension de la matrice de passage  $\phi$  associée et de l'architecture de la seconde couche du BCAM induite. En effet, en complément d'une matrice  $\phi$  de dimension  $((14 \times 14 \times 16) \times 4) \times 12\,912$  avec toutes les entrées composées de réels, l'architecture nécessite l'utilisation de 807 clusters comportant 16 neurones chacun et entraînant la création d'une matrice d'adjacence de dimension 12

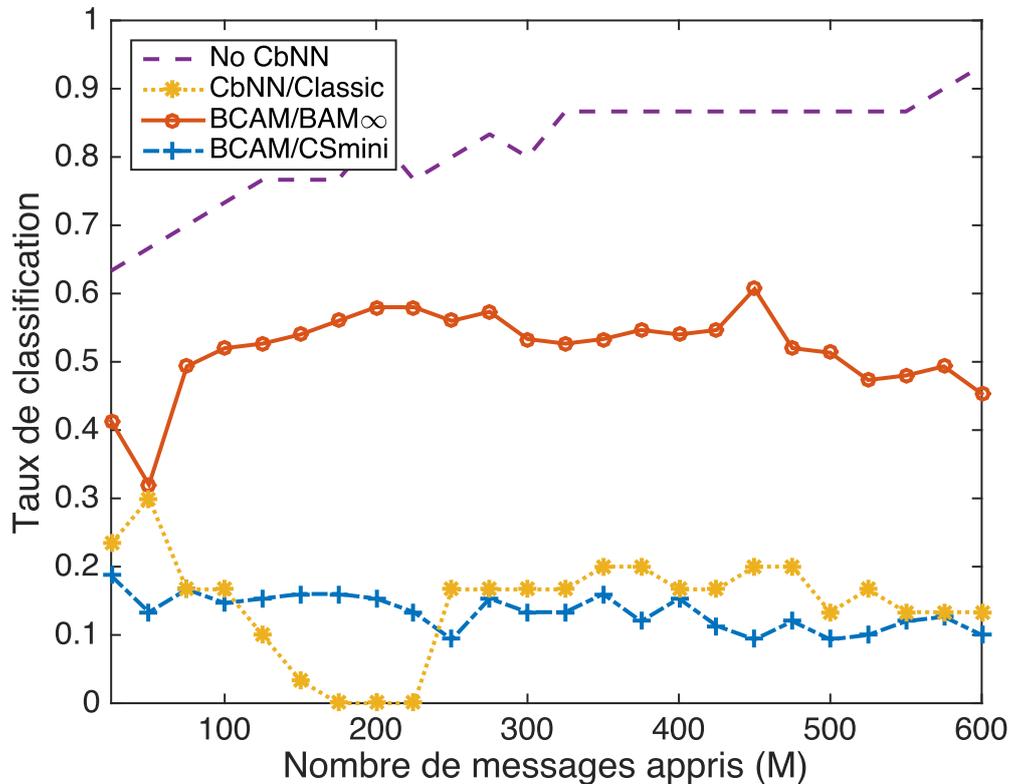
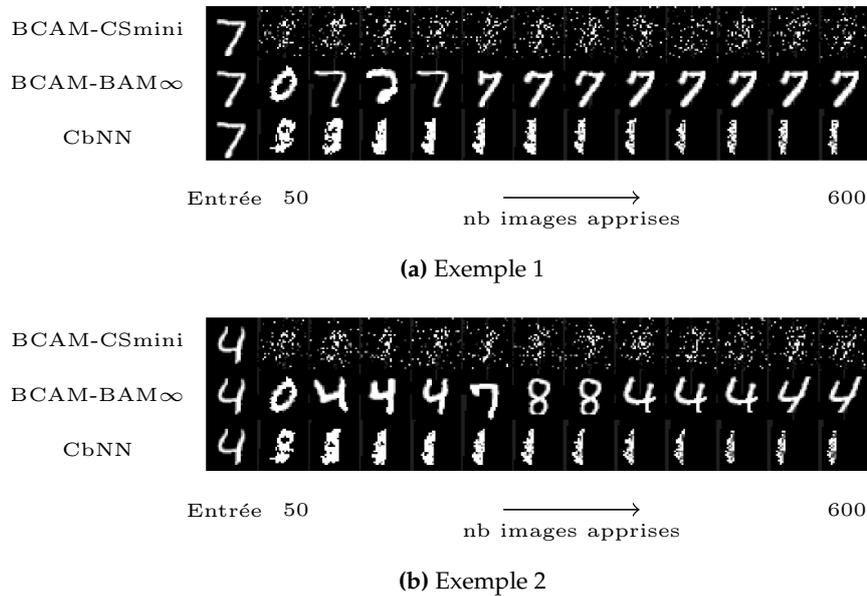


FIGURE 5.10 – Résultats de classifications des mémoires associatives à base de clique sur MNIST.

$912 \times 12\,912$ . L'utilisation de la méthode CS par parties diminue la quantité de mémoire nécessaire à travers la division de la matrice de passage, entraînant alors une réduction de dimension des sous matrices  $\phi^{(k)}$  associées aux sous-réseaux. La mémoire utilisée pour la méthode CS par parties reste élevée de par l'architecture de la seconde couche similaire à celle de la méthode CS pleine due à la dimension identique des macro-cliques quelle que soit la méthode CS. En dépit d'une matrice de passage de dimension plus élevée pour la méthode  $BAM_\infty$ , sa parcimonie induite par l'architecture de la seconde couche entraîne un faible coût mémoire. La mémoire utilisée à travers la méthode  $BAM_p$  est moindre comparée au CbNN original. En effet, cette réduction est imputable au faible nombre de clusters et neurones dans la seconde couche du BCAM et aux dimensions réduites des matrices d'adjacence des sous-réseaux par rapport à celle du modèle initial.

La figure 5.10 présente les résultats de classification du BCAM avec l'utilisation des méthodes CS par parties (à matrice de passage  $\phi$  unique) et  $BAM_\infty$ . Ces résultats sont comparés à ceux obtenus par le CbNN initial et sont donnés en fonction du nombre de messages appris. La classification est réalisée à l'aide de la méthode Softmax sur les images quantifiées de l'ensemble de test de MNIST, *c.-à-d.* des données non apprises par les mémoires associatives. L'entraînement du Softmax est effectué progressivement sur les mêmes images que



**FIGURE 5.11** – Exemples de reconstructions à l'aide du BCAM.

celles apprises par les réseaux à cliques. Afin de ne pas limiter la récupération du BCAM, une particularité de l'expérimentation avec des données inconnues est qu'elles ne sont pas corrigées par la première couche. En effet, si les images non apprises passent durant la première itération à travers la correction de la première couche, la récupération ratée sera diffusée au reste du réseau à travers la matrice de passage. Par conséquent, la seconde couche serait incapable de retrouver une macro-clique empêchant le réseau de récupérer une image de sa mémoire.

Du fait de son défaut de généralisation, le réseau CbNN est incapable de corriger des images non apprises, entraînant un taux de classification faible. Le réseau BCAM-CS obtient des résultats similaires à cause des mécaniques propres au CbNN. En effet, le facteur limitant de la méthode CS sur les données inconnues est la seconde couche. L'architecture de cette dernière étant pleine et étant donné le nombre élevé de clusters, la reconstruction sur des données non apprises est inefficace par les mécaniques de récupération du CbNN. En effet, l'utilisation de la méthode CS sur des données inconnues à l'aide d'une matrice  $\phi$  suivant une distribution gaussienne génère une macro-clique inconnue et donc impossible à corriger. Par la suite, la reconstruction en micro-clique à l'aide de l'algorithme BIHT est incorrecte à l'instar de l'échec de reconstruction sur la seconde couche. En ce qui concerne la méthode BAM $\infty$ , la matrice de passage étant parcimonieuse, la diffusion d'erreur est limitée puis corrigée par la seconde couche. En effet, plus la seconde couche est parcimonieuse, et plus la matrice d'association  $Z$  est parcimonieuse, ce qui réduit la diffusion d'erreurs.

La figure 5.10 démontre une tendance de généralisation du réseau jusqu'à un certain nombre de messages avant de chuter, en l'occurrence à 300 images apprises. Cette diminu-

tion de performances correspond à la valeur limite de stockage des sous-réseaux pour des données non-uniformes. La limite de densité pour des données artificielles non-uniformes est fixée à  $\delta = 0,2$  comme pour les résultats de la section (3.3.1). Nous avons observé que, pour les images de MNIST, plus la densité tend vers 0,1 et plus les ambiguïtés apparaissent, réduisant d'autant la capacité correctrice des CbNN.

La méthode est non supervisée expliquant des résultats sous-optimaux. La figure 5.11 illustre les reconstructions de deux exemples d'images inconnues soumises au réseau BCAM en fonction du nombre de messages appris. La première image à gauche correspond à l'entrée soumise au réseau. Puis nous donnons de gauche à droite une reconstruction de l'entrée à partir de la mémoire du réseau tous les 50 messages, jusqu'à un total de 600 images apprises. Chaque ligne correspond à une méthode. La première est le BCAM-CS par parties, les reconstructions montrent la diffusion d'erreurs à travers les couches rendant impossible la convergence du BCAM vers une image apprise au préalable. La seconde ligne est le BCAM-BAM $\infty$  : au fur et à mesure de l'apprentissage, le réseau converge vers le bassin d'attraction créé par la matrice de passage en fonction de l'entrée inconnue. L'évolution de l'image du "4" démontre la généralisation réalisée en fonction de l'apprentissage. La dernière ligne correspond aux reconstructions du réseau CbNN initial.

Dans la section suivante, nous appliquons le BCAM-BAM $\infty$  à nouveau sur MNIST mais avec une architecture plus parcimonieuse et sur deux autres jeux de données d'images.

### 5.4.3 Autres jeux d'images

Nous présentons des résultats de classification sur deux autres jeux de données d'images, Cifar10 et Yale illustrés dans l'annexe B. Les deux ensembles sont des images  $32 \times 32$  en couleurs pour Cifar10, *c.-à-d.* dotées des trois canaux RVB, et en nuances de gris pour Yale. Cifar10 consiste en 50 000 images d'apprentissage et 10 000 de test appartenant à 10 classes de type d'animaux et de véhicules. Les images de la base de Yale consistent en 165 images en nuances de gris composées de 15 classes représentant chacune un visage distinct. Pour une classe donnée, les images représentent différentes expressions faciales d'une même personne.

Dans cette section, nous expérimentons la capacité de généralisation du BCAM-BAM $\infty$  sur divers jeux de données d'images. Pour ces expérimentations, l'architecture de la seconde couche du BCAM-BAM est modifiée avec une la diminution du nombre de clusters de 400 à  $\chi_2 = 300$ , mais en ajoutant des neurones pour chaque cluster, passant de 32 à  $l_2 = 128$ . La modification provoque une hausse de la dimension de la matrice d'association  $Z$  afin d'accroître sa parcimonie. Le nombre de clusters actifs est désormais  $c_2 = 20$ . Les images sont découpées en  $p = 4$  patchs sans chevauchement de résolution  $16 \times 16$  chacun pour Cifar10 et Yale. Les résultats de classification sont les moyennes sur 200 (pour Cifar10) et 65

(pour Yale) images de test inconnues sur 4 séries, et ce pour 10 batches avec les sets d’images d’apprentissage et de test différents.

La figure 5.12 présente les résultats de classification à gauche et les densités associées aux couches des réseaux à droite. Nous avons testé à nouveau MNIST avec la nouvelle architecture sur l’ensemble des images de test, soit 10 000. La parcimonie de  $Z$  permet d’obtenir une amélioration en termes de classification. En effet, la figure 5.10 montrait que les performances étaient limitées à 0,60 de succès, quand désormais un taux de 0,65 est atteint. En outre, après l’apprentissage de 300 images, les résultats de classification ne sont pas dégradés. La densité associée à l’apprentissage des données MNIST, illustrée par la figure 5.12b, est faible du fait de la parcimonie des images.

Les résultats de Cifar10 sont présentés dans la figure 5.12c. Le faible taux de classification peut être expliqué par plusieurs raisons. Tout d’abord, nous sommes confrontés à une sur-quantification des images d’entrée et par conséquent à un biais important. En effet, les images codées sur les trois canaux RVB sont recodées en nuances de gris, puis quantifiées à l’aide de la méthode des K-moyennes. A la différence des images MNIST, celles de Cifar10 ne sont pas parcimonieuses. Par conséquent, elles nécessitent donc l’usage d’architectures pleines pour les sous-réseaux de la première couche du BCAM et l’ensemble du réseau CbNN initial. L’utilisation d’un réseau plein est le point faible des réseaux à clique comme illustré par les résultats de la méthode d’acquisition comprimée. En effet, les images pleines entraînent un remplissage rapide de la première couche du BCAM et du CbNN. Ce remplissage limite le nombre maximal d’images apprises à 300 avant que la densité ne tende vers 1 comme illustrée par la figure 5.12d. Les images sont des données non-uniformes et le seuil limite d’une récupération avec une probabilité faible d’obtenir des ambiguïtés est de  $\delta = 0,2$ . La surcharge du réseau et une perte importante de données sont les raisons des faibles performances de classification sur Cifar10. Une alternative serait un codage 8 bits des couleurs, mais cela nécessite un nombre excessif de neurones dans la première couche pour la consommation mémoire, sauf dans le cas d’une quantification supplémentaire. Une solution est l’exploitation des canaux par le BCAM. Chaque canal pourrait être géré par un BCAM avec deux couches, puis nous ajoutons une troisième couche afin de connecter l’ensemble des informations extraites par chaque sous-BCAM.

La figure 5.12e présente les résultats de classification sur les données de la base de visages Yale. Nous constatons comme pour Cifar10 que les réseaux sont pleins. Cependant, la perte d’information due à une quantification des nuances de gris étant moindre que celle observée sur les couleurs de Cifar10, les performances s’en trouvent améliorées. Le nombre d’images dans le jeu de données de Yale étant faible, la densité illustrée dans la figure 5.12f est réduite, et ce même avec l’utilisation d’architectures pleines. En outre, le faible nombre d’images nécessite d’utiliser des groupes. Ces groupes, au nombre de 100, consistent à sélectionner aléatoirement les images d’apprentissage, et les images restantes (inconnues du réseau) sont

utilisées pour mesurer la capacité de généralisation. Les résultats sur les images de Yale sont meilleurs que ceux sur MNIST, en dépit d'une plus large densité s'expliquant par une variance plus élevée entre les images de classes différentes.

La figure 5.13 présente des résultats complémentaires aux taux de classification. En effet, la figure 5.13a correspond à l'évolution des erreurs quadratiques moyennes (ou mean square error, MSE) entre l'entrée inconnue soumise au réseau et l'image produite telle que,

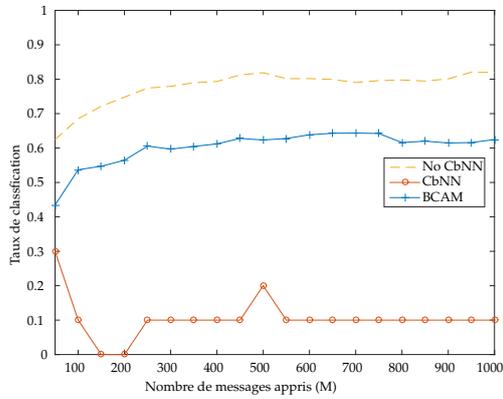
$$MSE(\hat{X}_i, X_i) = \frac{1}{n} \sum_{i=1}^n (\hat{X}_i - X_i)^2. \quad (5.16)$$

Par conséquent, plus le MSE diminue et meilleurs sont les résultats de classification. Pour le réseau BCAM, les erreurs associées aux différents jeux de données tendent à diminuer, contrairement au CbNN, illustrant la capacité généralisatrice du réseau bidirectionnel, qui cherche à approcher l'image soumise à l'aide de ses connaissances.

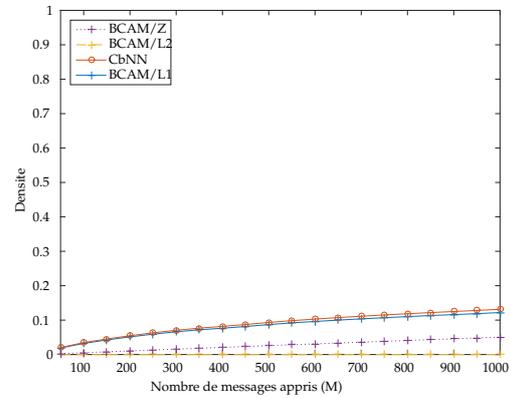
Deux autres indices de performance sont la consommation de la mémoire et le temps de calcul, respectivement illustrés par les figures 5.13b et 5.13c. En ce qui concerne le premier indice de performance, les données nécessitent une architecture pleine et par conséquent une consommation élevée de la mémoire. Cet accroissement de mémoire est plus important pour le CbNN puisqu'il enregistre toutes les connexions des neurones, tandis que l'architecture parcimonieuse de la première couche du BCAM réduit quadratiquement le nombre de connexions, et par conséquent le besoin en mémoire. En dépit d'un nombre de clusters et neurones élevé pour la seconde couche de BCAM nécessitant une matrice d'adjacence de haute dimension, la forte parcimonie du BCAM amène une consommation mémoire toujours plus faible que celle du CbNN.

Le second critère est le temps de calcul. La figure 5.13c illustre le temps moyen pour la reconstruction d'une image. Les expérimentations sont réalisées sur un processeur d'ordinateur de bureau i7 2600k cadencé à 3,4Ghz (2011). Cependant, le temps de récupération d'une image reste inférieur à la seconde, avec un maximum de 0,4 secondes pour un processeur i5 cadencé à 1,8Ghz d'un Macbook Air (2012). Nous pouvons faire l'hypothèse qu'un Raspberry Pi 2 serait en mesure de réaliser la récupération d'une image en moins d'une seconde. L'architecture et la parcimonie du BCAM requiert moins de temps de calcul que le CbNN originel. En effet, le nombre de connexions à traiter est inférieur pour le BCAM. Par ailleurs, le temps de calcul peut être encore réduit grâce à la parallélisation du traitement des données par les sous-réseaux de la première couche du BCAM.

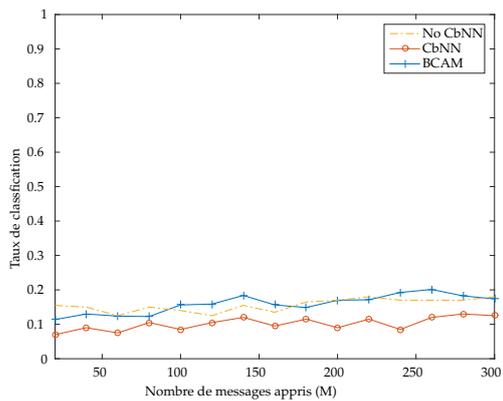
La figure 5.14 illustre les reconstructions produites par le BCAM (haut) et le CbNN (bas). Les exemples de Cifar10 démontrent que le BCAM ne retrouve pas nécessairement l'objet de la classe désirée, mais tend vers une forme semblable. Les exemples de visage illustrent la génération aléatoire des macro-cliques de la seconde couche par des visages qui n'ont aucun lien, mais qui restent produits par la mémoire. Les effacements sur les reconstructions du CbNN sont issus des ambiguïtés rencontrées. En effet, plusieurs neurones se retrouvent



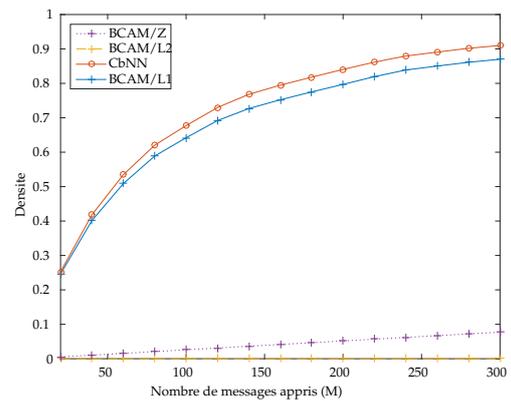
(a) Classification sur MNIST



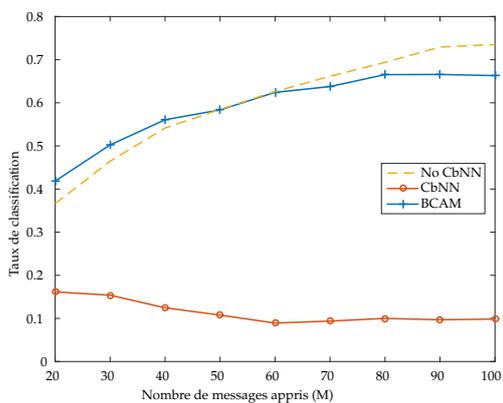
(b) Densité sur MNIST



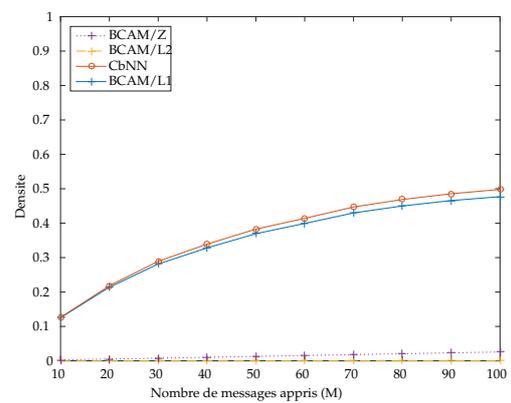
(c) Classification sur Cifar10



(d) Densité sur Cifar10

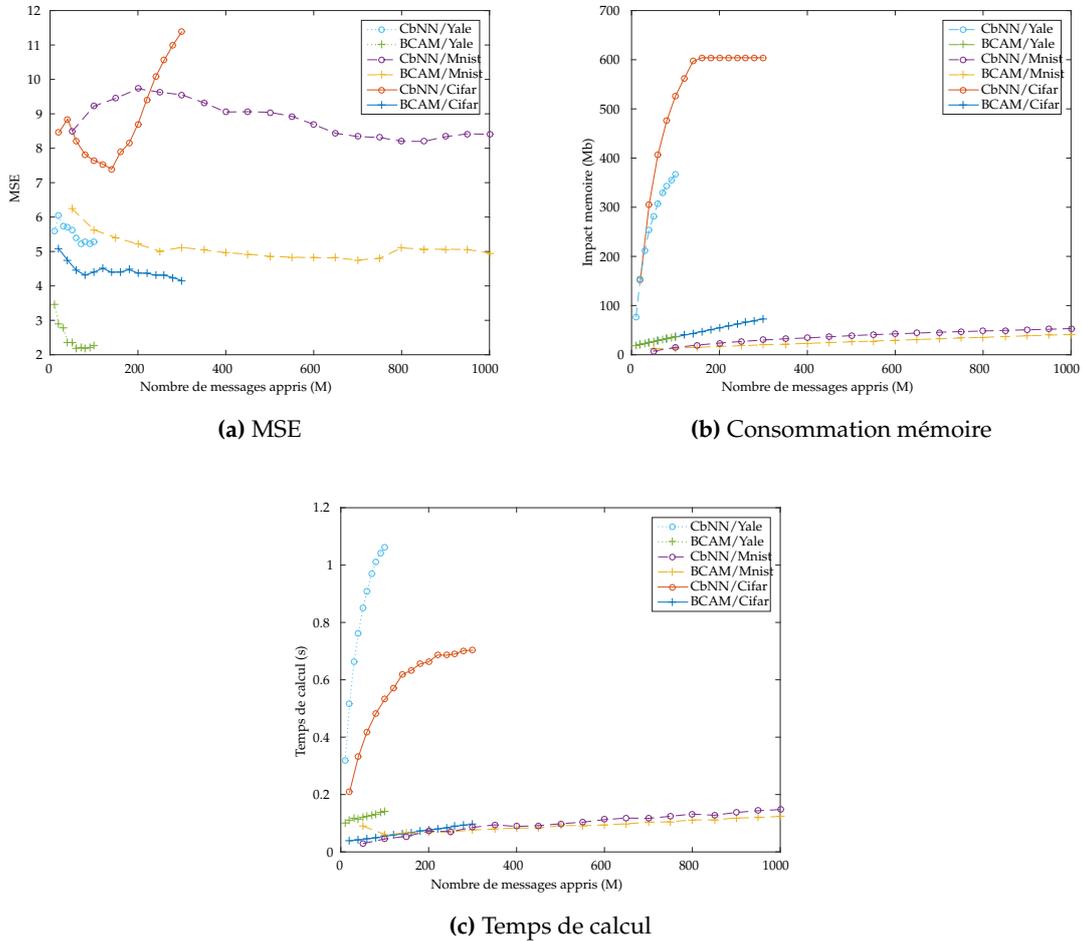


(e) Classification sur Yale



(f) Densité sur Yale

**FIGURE 5.12** – Taux de classification des réseaux à clique comparés à un Softmax sur des images inconnues en fonction du nombre d’images apprises, et les densités des réseaux utilisés.



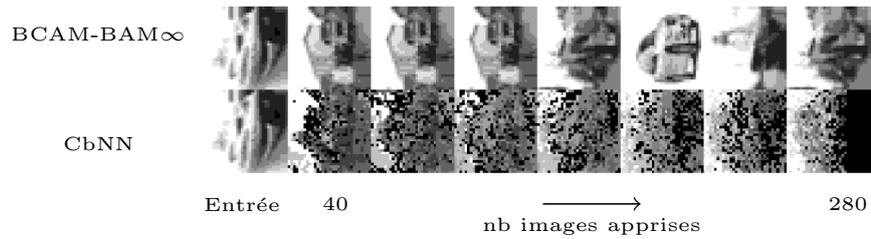
**FIGURE 5.13** – Indices de performances complémentaires des réseaux à clique comparé à un Softmax sur des images inconnues en fonction du nombre de messages appris.

activés dans un même cluster, et pour l’affichage des reconstructions nous appliquons un filtrage à l’aide de la règle d’activation WTA pour activer au maximum un neurone par cluster. Le filtrage provoque un effacement sur les figures d’images reconstruites.

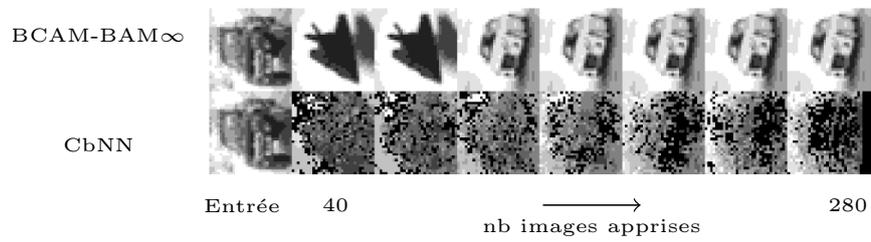
Quelles que soient les données traitées, le BCAM est en mesure de réaliser une meilleure correction, est capable de généralisation, et ce avec une utilisation minimale de mémoire entraînant un temps de récupération réduit.

## 5.5 Discussion sur la généralisation

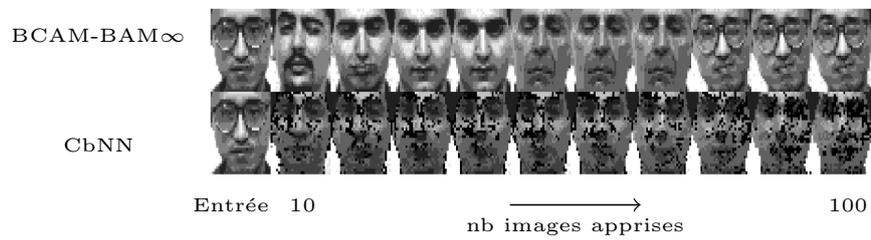
La généralisation du modèle BCAM-BAM $\infty$  est induite par la matrice d’association parcimonieuse. Un résultat complémentaire des mesures d’erreur MSE consiste en la distance



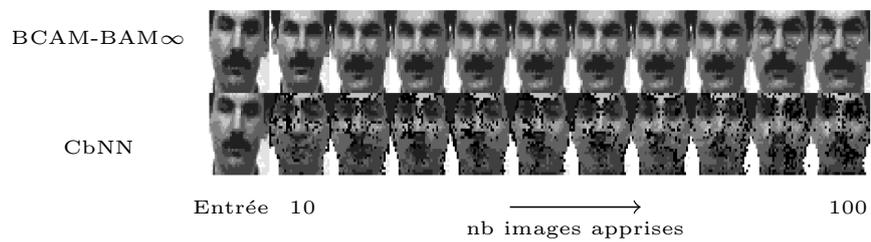
(a) Cifar10 Exemple 1



(b) Cifar 10 Exemple 2



(c) Yale Exemple 1



(d) Yale Exemple 2

FIGURE 5.14 – Exemples de reconstruction à l'aide des réseaux à cliques.

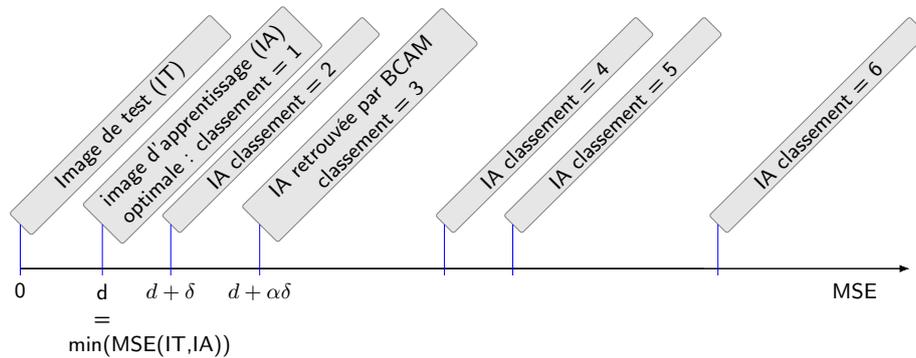
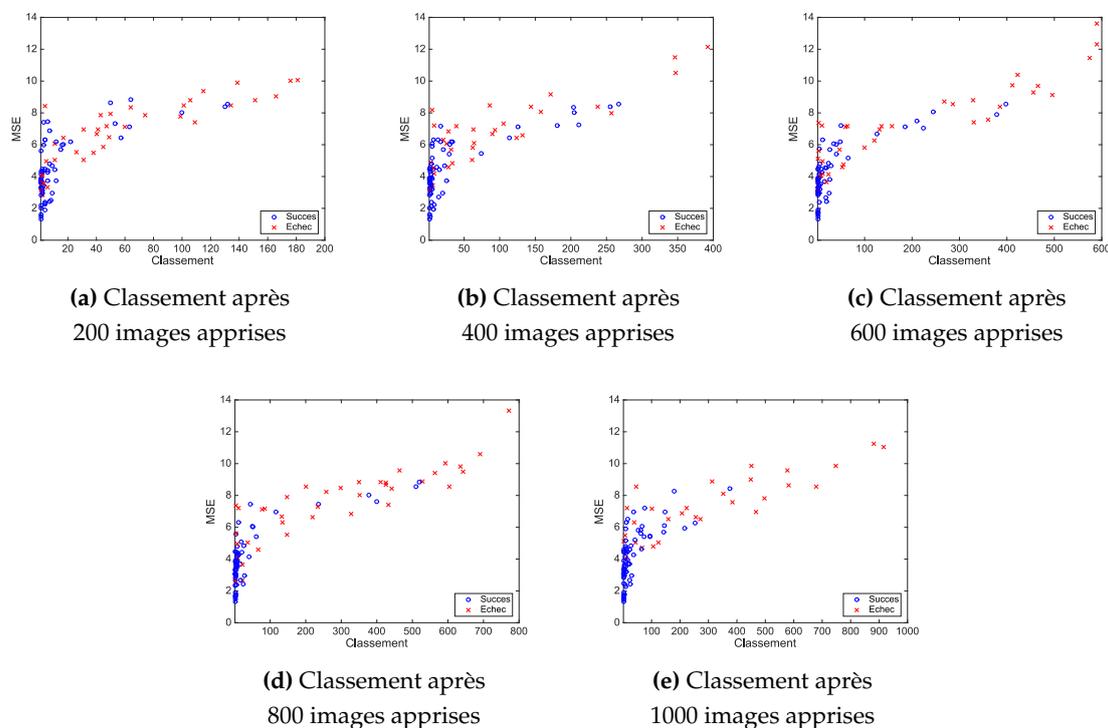


FIGURE 5.15 – Calcul des classements.

du résultat produit par le BCAM avec le résultat optimal. En effet, nous déterminons quelle image apprise minimise l'erreur MSE par rapport à l'entrée inconnue soumise. Ensuite nous calculons, avec le critère MSE, la distance entre le résultat produit par le BCAM et l'ensemble des images apprises. Ainsi nous déterminons un classement par rapport à une distance (MSE) entre l'image récupérée et l'image optimale de l'ensemble d'apprentissage. Le calcul du classement est illustré par la figure 5.15. En reprenant le cas d'étude des images MNIST, la figure 5.16 présente ces classements après apprentissage de 200, 400, 600, 800 et enfin 1 000 images. Plus les points sont dans le coin en bas à gauche et meilleurs sont les classements, qui se caractérisent par une erreur MSE minimisée entre l'image récupérée par le BCAM et l'image optimale de l'ensemble d'apprentissage. Les points ronds correspondent aux succès de la classification à l'aide du classifieur Softmax. Au contraire, les croix rouges correspondent aux échecs de classification. La figure 5.16 illustre bien que le succès d'une classification est lié à une minimisation de l'erreur MSE par le BCAM. Les résultats sont issus d'une architecture de BCAM utilisée pour mesurer les taux de classification, soit  $\chi_2 = 300$  clusters dont  $c_2 = 20$  actifs, chacun composé de  $l_2 = 128$  neurones. Ainsi, la matrice d'association est de dimension  $2\,048 \times 38\,400$ .

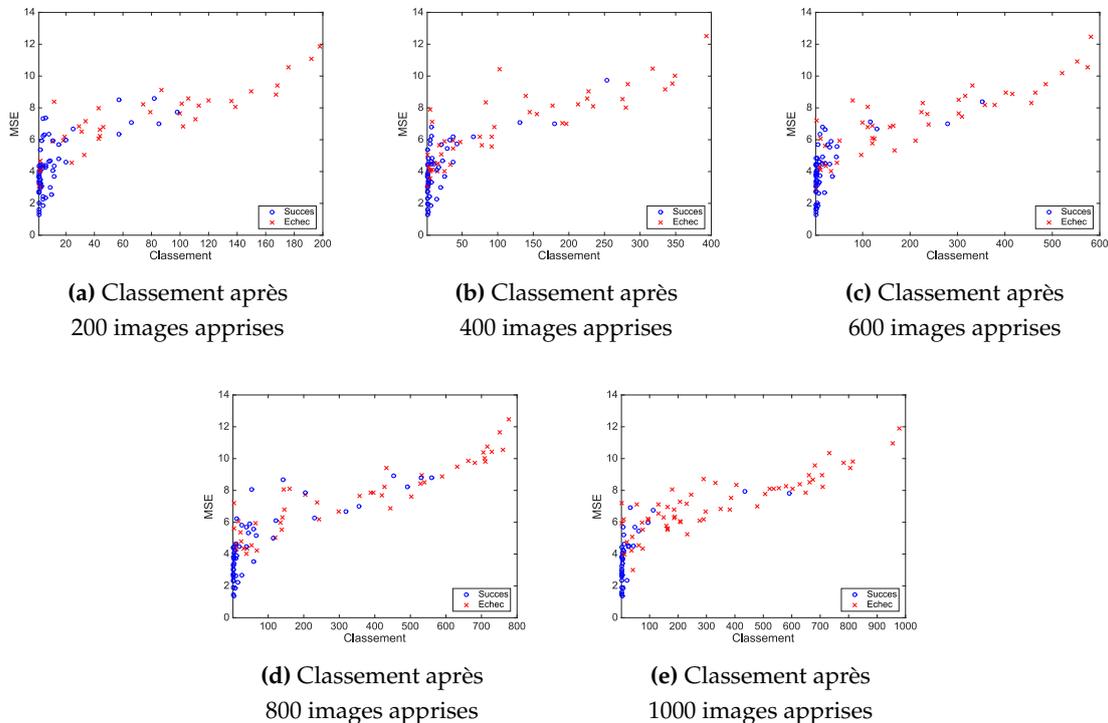
La figure 5.17 présente des résultats obtenus selon le protocole décrit précédemment, mais avec un réseau BCAM possédant une moindre parcimonie. La seconde couche comporte  $\chi_2 = 200$  clusters dont  $c_2 = 20$  actifs, chacun composé de  $l_2 = 32$  neurones. La dimension de la matrice d'association est désormais  $2\,048 \times 6\,400$ . Nous remarquons que le réseau BCAM n'est plus en mesure de classer aussi efficacement que le réseau plus parcimonieux. En effet, la faible parcimonie ne permet plus de maximiser la distance entre les macro-cliques et donc de les dissocier durant la phase de récupération. Le phénomène est illustré par l'accentuation du nombre d'échecs de classification (croix rouge) si un grand nombre d'images ont déjà été apprises. Les figures 5.16 et 5.17 illustrent l'approximation (selon le critère MSE) réalisée par le BCAM pour approcher au mieux une image inconnue avec une image apprise. Les performances d'approximation dépendent de la parcimonie de la matrice d'association. Par conséquent, grâce à la parcimonie, la mémoire associative bidirectionnelle



**FIGURE 5.16** – Classements et MSE avec un réseau à forte parcimonie comportant  $\chi_2 = 300$  clusters dont  $c_2 = 20$  actifs possédant chacun  $l_2 = 128$  neurones.

à clique est capable de généralisation.

Le point clé de la généralisation est la projection des données à travers la matrice d'association. La projection aléatoire est une méthode efficace utilisée notamment dans la réduction de dimension [16] [131]. En effet, d'après le lemme de Johnson et Lindenstrauss [79], les distances sont approximativement préservées après le passage des données à travers une matrice aléatoire de plus faible dimension, résultant en une simplification des calculs. L'utilisation de la méthode CS est une méthode semblable à la projection aléatoire à travers la matrice de passage  $\phi$  suivant une distribution non-uniforme. C'est aussi à cause de la conservation des distances et la réduction des dimension après la transition des données par la matrice  $\phi$  que la seconde couche rencontre des ambiguïtés, et n'est plus en mesure de récupérer l'information. Par conséquent, l'erreur se diffuse à travers le réseau comme illustré par les reconstructions de la figure 5.11. Cependant, dans notre cas, l'utilisation d'un tirage uniforme combiné à une forte parcimonie induite par la dimension souhaité du réseau permet d'étendre la distribution des neurones à activer et ainsi maximiser leurs dissociations durant la phase de récupération dans la seconde couche. La figure 5.18 illustre cette expansion. Nous pouvons comparer à gauche la distribution des données initiales (données artificielles pour cause de dimension), et à droite la distribution générée aléatoirement pour les macro-



**FIGURE 5.17** – Classements et MSE avec un réseau à faible parcimonie comportant  $\chi_2 = 200$  clusters dont  $c_2 = 20$  actifs possédant chacun  $l_2 = 32$  neurones.

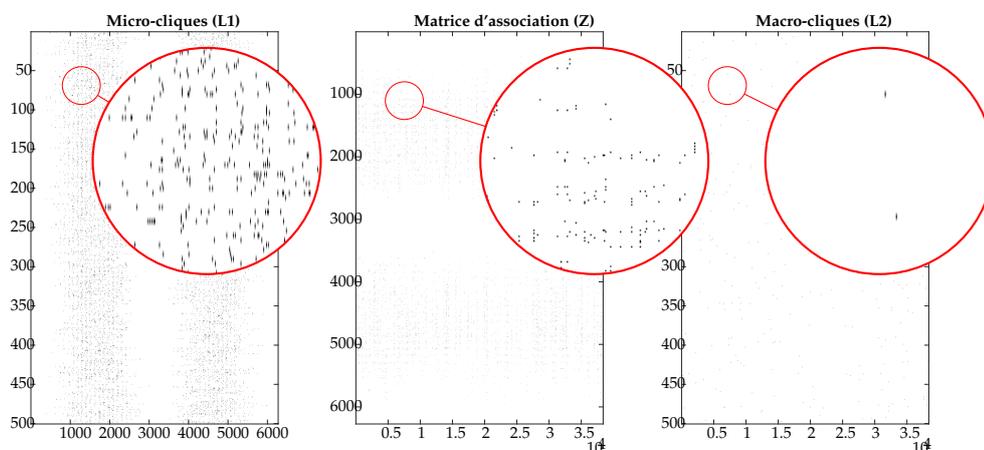
cliques. La matrice d’association résulte du produit des deux distributions, et la figure au centre illustre l’extension de la distribution pour la distinction des neurones.

En conclusion, du fait d’un tirage uniforme, les neurones activés dans la seconde couche du réseau BCAM se retrouvent loin les uns des autres, contrairement à la distribution non-uniforme des images. Cette forme de maximisation de distance permet de réduire l’appariation des ambiguïtés, et par conséquent de corriger plus efficacement.

### 5.5.1 Étude de la dimension des patches

Les protocoles d’expérimentations du réseau BCAM s’appuient sur des données décomposées en 4 parties de dimensions égales. Cependant, la dimension des patches lors de la récupération d’images peut influencer sur la généralisation. En effet, en fonction des données utilisées, la décomposition est équivalente à une contrainte posée sur les caractéristiques sélectionnées. Ces caractéristiques sont représentées par leurs distributions au sein de la mémoire du réseau.

La figure 5.19 illustre l’influence de la dimension des patches (sans chevauchement) sur les taux de classification. Les expérimentations sont réalisées sur les images du jeu de données



**FIGURE 5.18** – Distributions de l’activation neuronale des micro-cliques (à gauche); Matrice d’association (au milieu); Distributions de l’activation neuronale des macro-cliques (à droite).

MNIST avec le BCAM-BAM $\infty$ . Son architecture en première couche est la même que dans le tableau 5.2. La seconde couche comporte  $\chi_2 = 300$  clusters dont  $c_2 = 20$  sont actifs. Chaque cluster comporte  $l_2 = 128$  neurones. Chaque entrée de la matrice correspond au maximum du taux de classification obtenu à travers 4 séries sur 150 images inconnues soumises au réseau après apprentissage progressif jusqu’à 400 images. L’entrée supérieure gauche de la matrice, illustrée par la figure 5.19, représente les résultats de classification avec des patches de dimension  $2 \times 2$  et, par conséquent, la première couche du BCAM est composée de  $p = 196$  sous-réseaux CbNN. Le coin inférieur gauche représente les résultats avec des patches de dimension  $28 \times 2$ , *c.-à-d.* des bandes verticales, où le réseau associé possède une première couche composée de  $p = 14$  sous-réseaux. La conclusion que nous pouvons tirer quant à la décomposition avec le jeu de données de MNIST est que des bandes verticales accomplissent une meilleure généralisation car elles offrent une plus large diversité de caractéristiques. Chaque jeu de données suit une distribution particulière et dispose ainsi de caractéristiques spécifiques. L’étude de la dimension des patches associés aux sous-réseaux de la première couche du BCAM est crucial afin d’optimiser la généralisation du réseau en l’adaptant à ces caractéristiques spécifiques.

## 5.6 Conclusion

Le CbNN est incapable de corriger sans erreur un message inconnu qui lui est soumis. En effet, la phase de correction consiste à activer les neurones communs à l’entrée inconnue et à la mémoire, entraînant une récupération de plusieurs cliques, et l’apparition de chevau-chements. Cette problématique est inhérente aux mécaniques de récupération, qui sont dans

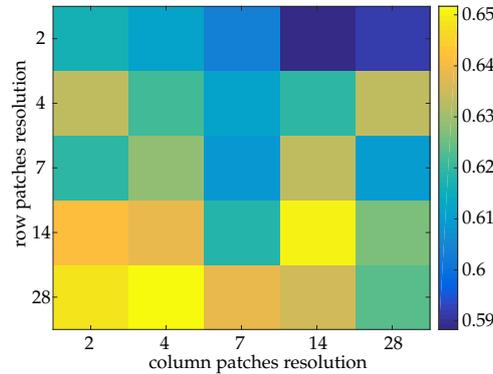


FIGURE 5.19 – Taux de classification en fonction de la dimension des patches.

l’incapacité de distinguer une clique d’une autre compte tenu de l’absence de contrainte globale sur le réseau. Autrement dit, la correction est réalisée du point de vue des neurones, soit localement.

Nous proposons une méthode permettant d’ajouter une contrainte globale en utilisant un réseau à deux couches qui s’occuperont respectivement de la correction d’un point de vue des neurones puis de la clique. Le passage d’une dimension à une autre est fait par une matrice de passage bidirectionnel afin de minimiser la perte d’information entre les couches. La capacité correctrice des CbNN de chaque couche permet de récupérer l’erreur diffusée par la matrice de passage jusqu’à un certain point. Le nouveau réseau de mémoire associative à cliques bidirectionnelle (BCAM) est dérivé selon plusieurs méthodes en fonction du traitement de l’information entre les couches.

Une première méthode BCAM-BAM s’appuie sur les mémoires associatives bidirectionnelles (BAM) afin de faire transiter l’information d’une couche à une autre et inversement. Ces mémoires BAM sont modifiées afin de limiter le coût mémoire dans l’ensemble du réseau BCAM. La réduction apporte une dégradation de l’information et est contrebalancée par la correction des couches de CbNN. L’idée initiale consistant à associer le nombre de clusters en seconde couche au nombre de patches (représentant une décomposition des données) a été étendue à un nombre plus élevé de clusters dans  $L_2$ . Cette extension maximise la parcimonie de la seconde couche et de la matrice de passage associée. En effet, la performance du CbNN dépend de sa parcimonie. La capacité de généralisation du BCAM est également assurée par la parcimonie de la matrice d’association qui permet de maximiser la distance entre les macro-cliques.

Une limite du réseau BCAM-BAM est la génération de macro-cliques sur la base d’une activation aléatoire des neurones dans la couche supérieure. Notre solution est d’utiliser la méthode d’acquisition comprimée (CS) afin d’extraire les caractéristiques des micro-cliques à l’aide d’une matrice aléatoire  $\phi$ . Le passage retour est possible à haute probabilité suivant

les conditions de parcimonie des données d'entrée et de l'incohérence entre les micro-cliques et macro-cliques obtenues par la matrice aléatoire. Le réseau BCAM-CS est malheureusement limité dans son pouvoir de généralisation en raison du manque de parcimonie au sein de sa matrice de passage, impactant l'architecture de la seconde couche devenue le point faible de cette méthode. Cependant, l'utilisation de BCAM-CS se révèle intéressante sur des données pour une tâche de correction sur des données connues mais non uniformes. Au final, sur des données artificielles, le BCAM-BAM $\infty$  reste la méthode la plus efficace, et ce même parmi les méthodes de l'état de l'art du CbNN.

L'utilisation des images MNIST, Cifar10 et Yale indique que le BCAM tend à la généralisation si les données d'entrée sont parcimonieuses et centrées afin de profiter pleinement de l'architecture du BCAM. Notre contribution a consisté en l'introduction d'un nouveau réseau de mémoire associative, le BCAM-BAM. Celui-ci corrige mieux que le réseau CbNN, et de plus il est capable de généraliser, tout en possédant une consommation mémoire minimale, et par conséquent un temps de récupération réduit. Cependant, les expérimentations sur Cifar10 indiquent que le réseau BCAM n'est pas encore apte à gérer des images couleurs.

En outre, le réseau offre un riche potentiel d'évolution. En effet, il est important de noter que les résultats reposent sur une méthode d'apprentissage non supervisée. Une évolution importante qui permettrait d'accroître les performances de généralisation consisterait à superviser l'apprentissage en maximisant la distance des macro-cliques si elles sont de classes différentes, et minimiser la distance si elles sont de même classe.

Une autre évolution concerne la plasticité du réseau. Dans notre cas, le réseau est limité à deux couches, mais il est extensible afin d'extraire toujours plus de caractéristiques tel un réseau de neurones profond dans une optique de *deep learning*.



# Chapitre 6

## Conclusion

---

### Contenu

<b>6.1 Contributions</b>	<b>132</b>
6.1.1 Ambiguïtés	133
6.1.2 Généralisation	134
<b>6.2 Perspectives</b>	<b>136</b>

---

Dans cette thèse, nous avons conduit l'étude d'une mémoire associative neuronale et son codage parcimonieux pour des tâches de vision par ordinateur. Les réseaux de neurones actuels reposent sur une consommation massive de ressources afin d'optimiser les poids de connexions entre les neurones. Cette contrainte est amplifiée par le développement de réseaux avec toujours plus de couches empilées, entraînant l'optimisation sur des centaines de millions de neurones.

Selon que le modèle soit discriminatif ou génératif, l'optimisation nécessitera respectivement la rétro-propagation ou la divergence contrastive. Ces méthodes correspondent à la manière de calculer les dérivées partielles des poids et biais du réseau en fonction des entrées afin de satisfaire la fonction de coût du réseau, *c.-à-d.* à minimiser l'erreur entre la sortie des neurones et celle désirée. La minimisation est conduite par une descente de gradient. Les ressources nécessaires afin de satisfaire un apprentissage à l'aide de ces méthodes sont du temps et une forte puissance de calcul générée par les cartes graphiques dont les puissances n'ont cessé d'évoluer ces dernières années.

Les performances des réseaux de neurones à travers l'avènement du *deep learning*, qui consiste en l'empilement de couches successives de neurones, ont provoqué le développement de technologies à destination des particuliers telles que la traduction d'une langue à

une autre en temps réel de panneaux d'affichage, la détection de véhicules pour les feux de signalisation, et les véhicules autonomes.

La généralisation de cette technologie entraîne un besoin d'extension à des appareils disposant de moins de ressources qu'une carte graphique, par exemple les smartphones. De ce fait, une des orientations des travaux actuels sur les réseaux de neurones cherche à réduire le coût des ressources afin d'exporter l'apprentissage automatique à toutes les formes de supports telles que les puces à faible ressources.

Nous avons vu qu'il est plus simple, mais pas nécessairement plus efficace, de stocker directement les informations en mémoire que de les traiter, d'où les réseaux de neurones sous forme de mémoire associatives. Autrement dit, les réseaux optimisés nécessitent un temps de calcul afin de traiter et modifier l'information pour satisfaire un besoin de classification ou reconnaissance. En revanche, les mémoires associatives stockent directement les informations, et leurs mécanismes de récupération cherchent à approcher les entrées soumises en fonction de leurs connaissances apprises.

Au delà de l'orientation actuelle des recherches dans le domaine des réseaux de neurones, notre implication dans le projet SENSE dont le but est d'imiter le processus de vision par un ordinateur à faible consommation de ressources justifie nos travaux sur les mémoires associatives neuronales. En outre, nous avons choisi d'utiliser les réseaux à cliques CbNN plutôt que d'autres mémoires associatives du fait de leurs architectures parcimonieuses qui, à nombre de neurones égal, offrent une plus large diversité. En effet, le CbNN est le réseau apprenant le plus d'informations avant de devenir saturé et incapable de corriger.

En revanche, une limite des CbNN, à l'instar des autres mémoires associatives, est qu'ils n'ont été validés que sur des données artificielles de faible dimension générées aléatoirement. Par conséquent, notre principal défi était d'adapter ces réseaux à des données telles que des images, *c.-à-d.* des données non uniformes et de haute dimension.

Nous résumons dans la section suivante les contributions apportées sur le réseau afin de l'aider à traiter des images ainsi que les perspectives qui s'ouvrent à partir des travaux que nous avons menés.

## 6.1 Contributions

Les contributions que nous avons apportées aux problématiques liées à l'utilisation de la mémoire associative CbNN s'articulent autour de deux axes majeurs : les *ambiguïtés* et la *généralisation*. En effet, l'utilisation de données non uniformes entraîne une même distribution dans la mémoire à travers leurs codages. Par conséquent, seul un sous-ensemble des neurones, support de l'information, est activé en présence de données non-uniformes. Outre le faible nombre de neurones partageant les informations apprises, le réseau est incapable de

réaliser une distinction. Cette problématique est appelée ambiguïté. La première partie de nos contributions a consisté à réduire les ambiguïtés afin de ne pas limiter les performances du réseau.

### 6.1.1 *Ambiguïtés*

Les performances et la validation du modèle initial s'appuient sur l'apprentissage de données artificielles uniformes. Lorsque le réseau est confronté à des données non uniformes, les contributions de la littérature cherchent à re-coder uniformément ces données à l'aide de la méthode du codage de Huffman [19] afin de maximiser les performances du CbNN. Une autre solution est la plasticité. Compte tenu d'une utilisation massive des mêmes neurones pour représenter les données non uniformes, une distinction des cliques peut s'accomplir en clonant ces neurones afin de partager le support de l'information et réduire les ambiguïtés [142].

Notre contribution repose également sur la plasticité afin de créer de nouvelles couches de CbNN et de partager les informations sur ces nouvelles couches et réduire ainsi l'apparition d'ambiguïtés. L'extension du modèle à N couches démontre une haute performance en terme de correction sur des données non uniformes. Cependant, les performances requièrent un coût en consommation mémoire et en temps de récupération des données linéaire et non plus logarithmique.

A défaut de plasticité, nous avons proposé une méthode de pénalisation des activations des neurones afin de ne les sélectionner que si et seulement si les neurones connectés forment une clique. La pénalisation est un processus postérieur à la règle dynamique consistant à calculer les scores des neurones. Les scores des neurones sont modifiés en fonction de leur appartenance à la clique associée à l'entrée soumise. La modification entraîne une sélection plus précise des neurones à activer afin de représenter l'information. La méthode se révèle efficace et peut faire disparaître définitivement les ambiguïtés du réseau selon la force de pénalisation soumise au réseau. La force nécessaire est analysée au même titre que la pénalisation. Les expérimentations pour valider la pénalisation reposent sur des données artificielles uniformes et gaussiennes, puis sur des images du jeu de données MNIST. Les données utilisées ont été apprises au préalable, puis dégradées avant une soumission pour correction. En effet, le CbNN n'est pas en mesure de généraliser et donc d'approcher une information inconnue par ce qu'il a déjà appris.

La deuxième partie des contributions repose sur l'aide apportée au réseau afin qu'il puisse généraliser sa mémoire à des données non apprises. En effet, la récupération d'une information inconnue par le CbNN entraîne l'activation des neurones de manière locale sans prendre en compte l'appartenance possible à une seule et même clique. La récupération résulte en un chevauchement de diverses cliques.

### 6.1.2 Généralisation

Notre première contribution a été de démontrer la limite inhérente à l'utilisation exclusive des mécaniques de récupération, rendant impossible la généralisation. A cet égard, nous utilisons une méthode semblable à la pénalisation afin de reconstruire une clique, mais suivant un a priori dépendant de l'entrée inconnue soumise. En effet, les activations des neurones sont contraintes par leurs distances avec les neurones qui devraient être activés en fonction de l'entrée. La méthode a pour effet d'activer les neurones relatifs à l'entrée inconnue sans prendre en compte l'appartenance à une unique clique, et en dépit de l'utilisation ultérieure d'une interpolation de la méthode d'a priori avec la pénalisation. La première conclusion est que les règles de récupération ne prennent pas en compte l'information globale du réseau, *c.-à-d.* une unique clique en entier, empêchant la reconstruction d'une clique approchée. La seconde conclusion est que les règles s'appuient sur l'architecture du CbNN : celle-ci devient le point faible du réseau avec des données inconnues.

La seconde contribution est une nouvelle architecture composée de deux couches de CbNN avec une matrice binaire bidirectionnelle afin de faire transiter l'information entre les couches. Le nom du modèle est *Mémoire Associative Bidirectionnelle à base de Clique* (ou Bidirectional Cliques based Associative Memory, BCAM). Une première couche enregistre les informations localement en associant un CbNN pour chaque partie des données découpées sous forme de patches. Le découpage des données entraîne une réduction des connexions dans la matrice d'adjacence associée aux sous-réseaux CbNN de la première couche. Ces sous-réseaux codent les micro-cliques. La seconde couche du BCAM code les connexions entre les sous-réseaux agissant comme une vue globale des informations codées localement. Les macro-cliques générées grâce à la seconde couche sont associées avec les micro-cliques de la première couche par une matrice de passage bidirectionnelle.

Outre l'utilisation de la matrice de passage comme une matrice d'association caractérisant la méthode BCAM-BAM qui repose sur le concept des mémoires associatives bidirectionnelles [88], nous proposons une méthode de réduction de dimension à base de *compressive sensing*, le BCAM-CS. En effet, l'utilisation du BCAM-BAM s'appuie sur une activation aléatoire des neurones de la seconde couche, tandis que pour le BCAM-CS, les neurones activés dépendent des caractéristiques extraites à travers une matrice aléatoire.

Les expérimentations réalisées avec le BCAM démontre de meilleures performances qu'avec le CbNN initial en termes de récupération sur des données apprises et dégradées, et ce quelle que soit la méthode utilisée. En dépit d'un nombre de neurones supérieur, la consommation de la mémoire pour le BCAM-BAM reste du même ordre que celle du CbNN initial. Cependant, l'utilisation d'une matrice aléatoire dans le BCAM-CS nécessite plus de ressources que les autres réseaux, y compris que le CbNN initial. En outre, le temps de récupération est réduit grâce à la réduction du nombre de connexions au sein du réseau BCAM.

L'utilisation d'images de dimension supérieure à des données artificielles démontre la limite du BCAM-CS compte tenu de sa consommation mémoire et de ses performances de récupération inhérentes à l'architecture pleine de la seconde couche. Cette limite empêche une récupération optimale de l'information globale et par conséquent rend impossible la généralisation d'images inconnues à travers le BCAM-CS. En dépit d'une activation aléatoire des neurones de la seconde couche, le BCAM-BAM obtient les meilleures performances, que ce soit en termes de correction, de consommation mémoire, de temps de récupération, et ce que les données soient connues ou non. En effet, le réseau est utilisé sur différents jeux d'images : MNIST, CIFAR10 et Yale.

Les performances de récupération sur des images inconnues permettent d'obtenir jusqu'à 70% de réussite de classification sur les visages de Yale. Les mauvaises performances sur CIFAR10 s'expliquent par une trop forte quantification des données afin de réduire leur dimension. En effet, CIFAR10 est composé d'images couleurs codée en 24 bits. L'apprentissage à l'aide du BCAM nécessite une première quantification en nuance de gris sur 8 bits puis une seconde à 4 bits. Cette perte de données ne permet pas au BCAM de généraliser. Bien que dans le cas de CIFAR10, le réseau soit incapable de converger vers une image apprise de la même classe, le BCAM récupère néanmoins une image similaire en terme de forme. La totalité des images de l'ensemble de test de MNIST a été traitée par le BCAM-BAM avec un taux de classification approchant les 70% de succès. Les performances sont en deçà de celles qu'atteignent les réseaux optimisés. Cependant, contrairement à ces derniers qui nécessitent des dizaines de minutes voire des heures pour obtenir une classification optimale, le BCAM nécessite moins de 1 seconde d'apprentissage non supervisé pour un temps de récupération d'une image du même ordre (temps issu d'expérimentations sur un MacBook de 2012). Enfin, l'architecture entièrement binaire du réseau possède une consommation mémoire significativement inférieure aux réseaux optimisés, permettant au BCAM d'être embarqué sur des architectures à très faibles ressources.

Outre la prise en compte des problématiques d'ambiguïté et de généralisation, nous avons proposé une méthode de classification d'images à l'aide des mécaniques de récupération du CbNN. La méthode s'appuie sur le codage des images ainsi que leurs labels associés. La classification à l'aide du CbNN sur des données dégradées par un effacement obtient de meilleures performances que le classificateur Softmax, si et seulement si les données ont été apprises au préalable. Dans le cas d'images inconnues, en dépit d'un échec lors de la récupération, la classification tend vers les 70% de succès sur le jeu de données MNIST.

Nous n'avons pas réussi à atteindre les performances de réseaux optimisés en nous appuyant uniquement sur des mémoires. Ce constat s'explique par la dimension des données. Chaque type d'approche possède ses forces et faiblesses, et l'un ne peut pas entièrement remplacer l'autre. La meilleure solution est probablement d'allier les deux notions comme le font les réseaux récurrents LSTM [38] [62] [126].

## 6.2 Perspectives

Dans cette section nous présentons quelques directions de recherche en continuité avec les contributions présentées dans la thèse. Le réseau BCAM possède un potentiel d'évolution à analyser. En effet, son inconvénient est la génération aléatoire des macro-cliques, autrement dit l'apprentissage est non supervisé. Une évolution du réseau serait l'activation des neurones de la seconde couche en fonction des données à travers une extraction de caractéristiques. Ainsi, dans le cas de la distinction de deux images, si celles-ci sont de la même classe, les macro-cliques associées devraient être similaires. En revanche, si les images sont de classes distinctes, les macro-cliques doivent être dissemblables. Par conséquent, il est nécessaire d'apporter une métrique au sein du réseau.

Par ailleurs, contrairement aux réseaux de Hopfield, la récupération d'informations du CbNN n'est pas mathématiquement définie. En effet, tandis que le modèle de Hopfield s'appuie sur une fonction de coût pour converger vers des bassins d'attraction, la convergence des cliques du CbNN n'est pas démontrée. Ce défaut est inhérent au manque de métriques au sein du réseau à clique. Par conséquent, la compréhension accrue du réseau passe nécessairement par l'étude formelle de la convergence des cliques.

Un autre axe de recherche sur le BCAM concerne son architecture. En effet, nous avons étudié dans cette thèse un réseau BCAM constitué de deux couches. Cependant, l'étude des images couleur, selon les trois canaux RVB, du jeu de données CIFAR10 a démontré les limites du réseau BCAM face à la quantification. Une évolution serait de considérer un modèle possédant une hiérarchie étendue afin de restreindre l'usage de la quantification, et par conséquent minimiser la perte d'information.

A l'instar du *deep belief net* utilisant une mémoire associative, le BCAM peut être intégré dans des réseaux de neurones optimisés. Par exemple, l'auto-encodeur est un réseau de neurones consistant à représenter l'information soumise sous forme de codage parcimonieux. En effet, la couche cachée du réseau correspond à un dictionnaire de caractéristiques extraites dont les combinaisons linéaires permettent de reconstruire les données soumises. Cependant, l'utilisation de données dégradées réduit les performances du réseau. L'intégration au sein de la couche cachée de l'auto-encodeur d'un BCAM pourrait être une solution afin d'utiliser sa capacité correctrice et d'approximation sur les activations de la couche cachée.

Une dernière perspective concerne le coût en ressources du BCAM. En effet, les expérimentations menées sur les images (données à haute dimension) ont démontré qu'il était possible d'embarquer le réseau sur des supports à faibles ressources. Cependant, à défaut d'implanter le BCAM sur un support existant, une étape supplémentaire serait le développement d'une architecture matérielle dédiée au BCAM, à l'instar de celle développée pour le CbNN.

# Annexes

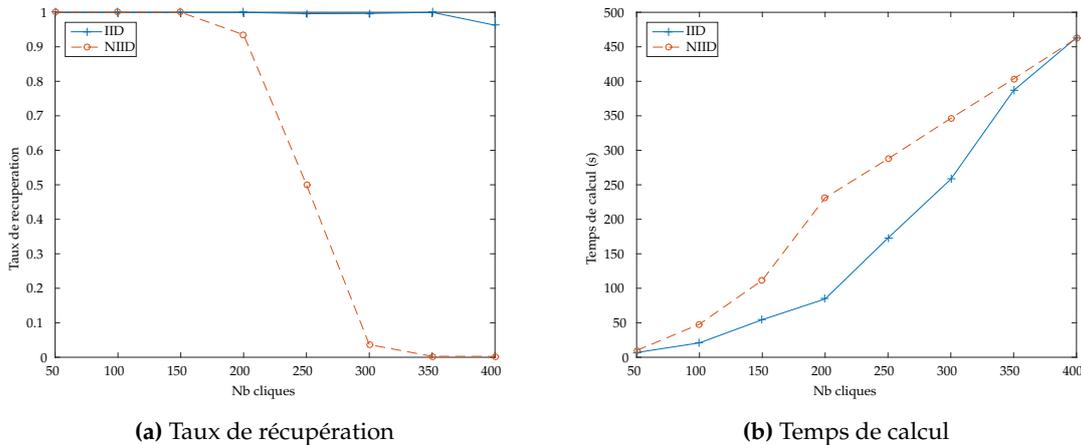


## A Factorisation de matrice

La factorisation de la matrice d’adjacence  $W$  correspondant au produit des cliques apprises  $T$  et sa transposée (équation 4.2) s’appuie sur la minimisation de l’équation

$$\min_{Q,P} \frac{1}{2} (\|A - Q^t P\|_1^2 + \|P - Q\|_1^2 + \|Q^2 - Q\|_1^2 + \|P^2 - P\|_1^2), \quad (1)$$

avec  $P$  égale à la matrice des cliques  $T$  (équation 3.2), et  $Q$  sa transposée. Le premier terme minimise l’erreur quadratique entre la matrice soumise à optimisation et celle reconstruite par minimisation. Le deuxième terme s’assure que la décomposition soit le produit d’une matrice et sa transposée,  $P_i \approx Q_j$ . Le troisième et quatrième termes accentuent la contrainte sur la binarité des valeurs des matrices  $Q$  et  $P$ . Les termes sont de normes  $L1$  pour obtenir de l’optimisation une solution parcimonieuse. Bien que la méthode fonctionne, elle est limitée par le nombre de messages appris. En effet, plus ce nombre est grand et plus le temps nécessaire à l’optimisation est accru comme illustré par la figure A.1b.



**FIGURE A.1** – Résultats de récupération de l’ensemble des cliques apprises à partir d’une matrice d’adjacence via une factorisation de matrice.

La figure A.1a présente les taux de récupération des cliques à partir d’une matrice d’adjacence entière en fonction de la distributions des activations neuronales (iid : uniforme et niid : gaussienne). Si le nombre de messages appris reste faible (permettant à l’optimisation de converger avant le nombre maximal d’itérations fixé à 600), l’ensemble des cliques peuvent être retrouvées sans erreur. Cependant, en fonction de la distribution des activations, la récupération à l’aide de l’optimisation souffre des mêmes défauts que les mécaniques propres au CbNN. Autrement dit, la distribution gaussienne des activations neuronales rend difficile la distinction des cliques. La méthode d’optimisation utilisée est la quasi-Newtonienne Broyden–Fletcher–Goldfarb–Shannon dans sa version limitant l’exploitation

de la mémoire (L-BFGS) [43]. Les résultats sont obtenus à l'aide d'une architecture pleine composée de  $\chi = c = 16$  clusters comprenant chacun  $l = 32$  neurones. Les temps de calculs reposent sur l'usage d'un processeur i7 2600k cadencé à 3,4 Ghz.

## B Jeux de données d'images

Cette section décrit les jeux de données d'images utilisés dans les protocoles d'expérimentation de la thèse sur la correction et la généralisation à l'aide d'un réseau de neurones à clique.

Jeu de données	Résolution	Profondeur	Taille de l'ensemble d'apprentissage	Taille de l'ensemble de test	Nombre de Classes
MNIST	$28 \times 28$	8 bits	60 000	10 000	10
Yale	$32 \times 32$	8 bits	100	65	15
CIFAR10	$32 \times 32$	24 bits	50 000	10 000	10

### B.1 MNIST



(a) Apprentissage



(b) Test

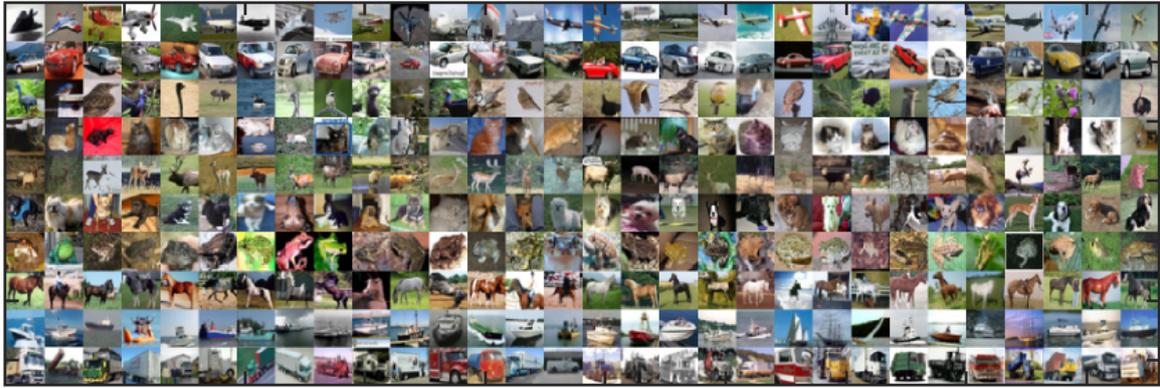
FIGURE B.1 – Visualisation des images du jeu de données MNIST.

## B.2 Yale



FIGURE B.2 – Visualisation de l’ensemble des images du jeu de données Yale.

### B.3 CIFAR10



(a) Apprentissage



(b) Test

**FIGURE B.3** – Visualisation des images du jeu de données CIFAR10.

# Bibliographie

---

- [1] Martin ABADI, Ashish AGARWAL, Paul BARHAM, Eugene BREVDO, Zhifeng CHEN, Craig CITRO, Greg S CORRADO, Andy DAVIS, Jeffrey DEAN, Matthieu DEVIN et al. "Tensorflow : Large-scale machine learning on heterogeneous distributed systems". In : *arXiv preprint arXiv :1603.04467* (2016).
- [2] Ala ABOUDIB, Vincent GRIPON et Xiaoran JIANG. "A study of retrieval algorithms of sparse messages in networks of neural cliques". In : *arXiv preprint arXiv :1308.4506* (2013).
- [3] David H ACKLEY, Geoffrey E HINTON et Terrence J SEJNOWSKI. "A learning algorithm for Boltzmann machines". In : *Cognitive science* 9.1 (1985), p. 147–169.
- [4] Behrooz Kamary ALIABADI, Claude BERROU, Vincent GRIPON et Xiaoran JIANG. "Storing sparse messages in networks of neural cliques". In : *IEEE Transactions on neural networks and learning systems* 25.5 (2014), p. 980–989.
- [5] Shun-Ichi AMARI. "Learning patterns and pattern sequences by self-organizing nets of threshold elements". In : *Computers, IEEE Transactions on* 100.11 (1972), p. 1197–1206.
- [6] Daniel J AMIT, Hanoch GUTFREUND et Haim SOMPOLINSKY. "Spin-glass models of neural networks". In : *Physical Review A* 32.2 (1985), p. 1007.
- [7] Daniel J AMIT, Hanoch GUTFREUND et Haim SOMPOLINSKY. "Storing infinite numbers of patterns in a spin-glass model of neural networks". In : *Physical Review Letters* 55.14 (1985), p. 1530.
- [8] James A ANDERSON. "A memory storage model utilizing spatial correlation functions". In : *Kybernetik* 5.3 (1968), p. 113–119.
- [9] James A ANDERSON, Jack W SILVERSTEIN, Stephen A RITZ et Randall S JONES. "Distinctive features, categorical perception, and probability learning : Some applications of a neural model." In : *Psychological Review* 84.5 (1977), p. 413.

- [10] Nicolas AUDEBERT, Bertrand Le SAUX et Sébastien LEFÈVRE. “Semantic Segmentation of Earth Observation Data Using Multimodal and Multi-scale Deep Networks”. In : *arXiv preprint arXiv :1609.06846* (2016).
- [11] Alan D BADDELEY. *Human memory : Theory and practice*. Psychology Press, 1997.
- [12] Frédéric BASTIEN, Pascal LAMBLIN, Razvan PASCANU, James BERGSTRÄ, Ian GOODFELLOW, Arnaud BERGERON, Nicolas BOUCHARD, David WARDE-FARLEY et Yoshua BENGIO. “Theano : new features and speed improvements”. In : *arXiv preprint arXiv :1211.5590* (2012).
- [13] Yoshua BENGIO, Patrice SIMARD et Paolo FRASCONI. “Learning long-term dependencies with gradient descent is difficult”. In : *IEEE transactions on neural networks* 5.2 (1994), p. 157–166.
- [14] James BERGSTRÄ, Olivier BREULEUX, Frédéric BASTIEN, Pascal LAMBLIN, Razvan PASCANU, Guillaume DESJARDINS, Joseph TURIAN, David WARDE-FARLEY et Yoshua BENGIO. “Theano : A CPU and GPU math compiler in Python”. In : *Proc. 9th Python in Science Conf.* 2010, p. 1–7.
- [15] Claude BERROU et Alain GLAVIEUX. “Turbo codes”. In : *Encyclopedia of Telecommunications* (2003).
- [16] Ella BINGHAM et Heikki MANNILA. “Random projection in dimensionality reduction : applications to image and text data”. In : *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, p. 245–250.
- [17] Christopher M BISHOP. “Pattern recognition”. In : *Machine Learning* 128 (2006), p. 1–58.
- [18] Thomas BLUMENSATH et Mike E DAVIES. “Iterative hard thresholding for compressed sensing”. In : *Applied and Computational Harmonic Analysis* 27.3 (2009), p. 265–274.
- [19] Bartosz BOGUSLAWSKI, Vincent GRIPON, Fabrice SEGUIN et Frédéric HEITZMANN. “Huffman coding for storing non-uniformly distributed messages in networks of neural cliques”. In : *AAAI 2014 : the 28th Conference on Artificial Intelligence*. T. 1. 2014, p. 262–268.
- [20] Bartosz BOGUSLAWSKI, Vincent GRIPON, Fabrice SEGUIN et Frédéric HEITZMANN. “Twin Neurons for Efficient Real-World Data Distribution in Networks of Neural Cliques : Applications in Power Management in Electronic Circuits”. In : *IEEE transactions on neural networks and learning systems* 27.2 (2016), p. 375–387.
- [21] Léon BOTTOU. “Large-scale machine learning with stochastic gradient descent”. In : *Proceedings of COMPSTAT’2010*. Springer, 2010, p. 177–186.
- [22] Léon BOTTOU. “Online learning and stochastic approximations”. In : *On-line learning in neural networks* 17.9 (1998), p. 142.

- [23] J BUHMANN, R DIVKO et K SCHULTEN. "Associative memory with high information content". In : *Physical Review A* 39.5 (1989), p. 2689.
- [24] Emmanuel J CANDÈS. "The restricted isometry property and its implications for compressed sensing". In : *Comptes Rendus Mathématique* 346.9-10 (2008), p. 589–592.
- [25] Emmanuel J CANDÈS et Justin ROMBERG. "Sparsity and incoherence in compressive sampling". In : *Inverse problems* 23.3 (2007), p. 969.
- [26] Emmanuel J CANDÈS et Michael B WAKIN. "An introduction to compressive sampling". In : *IEEE signal processing magazine* 25.2 (2008), p. 21–30.
- [27] Emmanuel J CANDÈS et al. "Compressive sampling". In : *Proceedings of the international congress of mathematicians*. T. 3. Madrid, Spain. 2006, p. 1433–1452.
- [28] Natalia CAPORALE et Yang DAN. "Spike timing-dependent plasticity : a Hebbian learning rule". In : *Annu. Rev. Neurosci.* 31 (2008), p. 25–46.
- [29] Adam COATES, Brody HUVAL, Tao WANG, David WU, Bryan CATANZARO et Ng ANDREW. "Deep learning with COTS HPC systems". In : *International Conference on Machine Learning*. 2013, p. 1337–1345.
- [30] Ronan COLLOBERT, Samy BENGIO et Johnny MARIÉTHOZ. *Torch : a modular machine learning software library*. Rapp. tech. Idiap, 2002.
- [31] Jerome T CONNOR, R Douglas MARTIN et Les E ATLAS. "Recurrent neural networks and robust time series prediction". In : *IEEE transactions on neural networks* 5.2 (1994), p. 240–254.
- [32] William K COULTER, Christopher J HILLAR, Guy ISLEY et Friedrich T SOMMER. "Adaptive compressed sensing—a new class of self-organizing coding models for neuroscience". In : *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE. 2010, p. 5494–5497.
- [33] Matthieu COURBARIAUX, Itay HUBARA, Daniel SOUDRY, Ran EL-YANIV et Yoshua BENGIO. "Binarized neural networks : Training deep neural networks with weights and activations constrained to  $\pm 1$ ". In : *arXiv preprint arXiv :1602.02830* (2016).
- [34] George E DAHL, Tara N SAINATH et Geoffrey E HINTON. "Improving deep neural networks for LVCSR using rectified linear units and dropout". In : *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, p. 8609–8613.
- [35] Robin DANILO, Philippe COUSSY, Laura CONDE-CANENCIA, Vincent GRIPON et Warren J GROSS. "Restricted clustered neural network for storing real data". In : *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM. 2015, p. 205–210.
- [36] Manolis DELAKIS et Christophe GARCIA. "text Detection with Convolutional Neural Networks." In : *VISAPP* (2). 2008, p. 290–294.

- [37] David DEMERS et GW COTTRELL. “n-linear dimensionality reduction”. In : *Adv. Neural Inform. Process. Sys* 5 (1993), p. 580–587.
- [38] Jeffrey DONAHUE, Lisa ANNE HENDRICKS, Sergio GUADARRAMA, Marcus ROHRBACH, Subhashini VENUGOPALAN, Kate SAENKO et Trevor DARRELL. “Long-term recurrent convolutional networks for visual recognition and description”. In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, p. 2625–2634.
- [39] David L DONOHO. “Compressed sensing”. In : *Information Theory, IEEE Transactions on* 52.4 (2006), p. 1289–1306.
- [40] David L DONOHO et Michael ELAD. “Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization”. In : *Proceedings of the National Academy of Sciences* 100.5 (2003), p. 2197–2202.
- [41] Jianfeng FENG, Mariya SHCHERBINA et Brunello TIROZZI. “On the critical capacity of the Hopfield model”. In : *Communications in Mathematical Physics* 216.1 (2001), p. 139–177.
- [42] Asja FISCHER et Christian IGEL. “An introduction to restricted Boltzmann machines”. In : *Iberoamerican Congress on Pattern Recognition*. Springer. 2012, p. 14–36.
- [43] Roger FLETCHER. *Practical methods of optimization*. John Wiley & Sons, 2013.
- [44] Peter FÖLDIÁK et Malcom P YOUNG. “Sparse coding in the primate cortex”. In : *The handbook of brain theory and neural networks* 1 (1995), p. 1064–1068.
- [45] D GABOR. “Associative holographic memories”. In : *IBM Journal of Research and Development* 13.2 (1969), p. 156–159.
- [46] Robert GALLAGER. “Low-density parity-check codes”. In : *IRE Transactions on information theory* 8.1 (1962), p. 21–28.
- [47] Christophe GARCIA et Manolis DELAKIS. “Convolutional face finder : A neural architecture for fast and robust face detection”. In : *IEEE Transactions on pattern analysis and machine intelligence* 26.11 (2004), p. 1408–1423.
- [48] Elizabeth GARDNER. “The space of interactions in neural network models”. In : *Journal of physics A : Mathematical and general* 21.1 (1988), p. 257.
- [49] Ehsan Sedgh GOOYA, Dominique PASTOR et Vincent GRIPON. “Automatic face recognition using SIFT and networks of tagged neural cliques”. In : *Proceedings of Cognitive*. 2015, p. 57–61.
- [50] Alex GRAVES, Abdel-rahman MOHAMED et Geoffrey HINTON. “Speech recognition with deep recurrent neural networks”. In : *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, p. 6645–6649.

- [51] Vincent GRIPON et Claude BERROU. “Nearly-optimal associative memories based on distributed constant weight codes”. In : *Information Theory and Applications Workshop (ITA), 2012*. IEEE. 2012, p. 269–273.
- [52] Vincent GRIPON et Claude BERROU. “Sparse neural networks with large learning diversity”. In : *IEEE Transactions on Neural Networks* 22.7 (2011), p. 1087–1096.
- [53] Vincent GRIPON, Judith HEUSEL, Matthias LÖWE et Franck VERMET. “A Comparative Study of Sparse Associative Memories”. In : *arXiv preprint arXiv :1512.08892* (2015).
- [54] Raia HADSELL, Pierre SERMANET, Jan BEN, Ayse ERKAN, Marco SCOFFIER, Koray KAVUKCUOGLU, Urs MULLER et Yann LECUN. “Learning long-range vision for autonomous off-road driving”. In : *Journal of Field Robotics* 26.2 (2009), p. 120–144.
- [55] Ugur HALICI. *Artificial Neural Networks, chap 3*. 2010.
- [56] Richard W HAMMING. “Error detecting and error correcting codes”. In : *Bell System technical journal* 29.2 (1950), p. 147–160.
- [57] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN. “Delving deep into rectifiers : Surpassing human-level performance on imagenet classification”. In : *Proceedings of the IEEE International Conference on Computer Vision*. 2015, p. 1026–1034.
- [58] Donald Olding HEBB. *The organization of behavior : A neuropsychological theory*. Psychology Press, 2005.
- [59] Christopher HILLAR, Ngoc TRAN et Kilian KOEPEL. “Robust exponential binary pattern storage in Little-Hopfield networks”. In : *arXiv preprint arXiv :1206.2081* (2012).
- [60] Geoffrey E HINTON. “Training products of experts by minimizing contrastive divergence”. In : *Neural computation* 14.8 (2002), p. 1771–1800.
- [61] Geoffrey E HINTON, Simon OSINDERO et Yee-Whye TEH. “A fast learning algorithm for deep belief nets”. In : *Neural computation* 18.7 (2006), p. 1527–1554.
- [62] Sepp HOCHREITER et Jürgen SCHMIDHUBER. “Long short-term memory”. In : *Neural computation* 9.8 (1997), p. 1735–1780.
- [63] John J HOPFIELD. “Neural networks and physical systems with emergent collective computational abilities”. In : *Proceedings of the national academy of sciences* 79.8 (1982), p. 2554–2558.
- [64] John J HOPFIELD. “Neurons with graded response have collective computational properties like those of two-state neurons”. In : *Proceedings of the national academy of sciences* 81.10 (1984), p. 3088–3092.
- [65] John J HOPFIELD et David W TANK. ““Neural” computation of decisions in optimization problems”. In : *Biological cybernetics* 52.3 (1985), p. 141–152.

- [66] Mark HOROWITZ. "1.1 Computing's energy problem (and what we can do about it)". In : *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE. 2014, p. 10–14.
- [67] Patrik O HOYER. "Non-negative matrix factorization with sparseness constraints". In : *The Journal of Machine Learning Research* 5 (2004), p. 1457–1469.
- [68] Patrik O HOYER et Aapo HYVÄRINEN. "A multi-layer sparse coding network learns contour coding from natural images". In : *Vision research* 42.12 (2002), p. 1593–1605.
- [69] Romain HUET, Nicolas COURTY et Sébastien LEFÈVRE. "A new penalisation term for image retrieval in clique neural networks". In : *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2016.
- [70] Niall HURLEY et Scott RICKARD. "Comparing measures of sparsity". In : *Information Theory, IEEE Transactions on* 55.10 (2009), p. 4723–4741.
- [71] Mark A IWEN. "Compressed sensing with sparse binary matrices : Instance optimal error guarantees in near-optimal time". In : *Journal of Complexity* 30.1 (2014), p. 1–15.
- [72] Laurent JACQUES, Jason N LASKA, Petros T BOUFONOS et Richard G BARANIUK. "Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors". In : *Information Theory, IEEE Transactions on* 59.4 (2013), p. 2082–2102.
- [73] Hooman JAROLLAHI, Vincent GRIPON, Naoya ONIZAWA et Warren J GROSS. "A low-power content-addressable memory based on clustered-sparse networks". In : *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*. IEEE. 2013, p. 305–308.
- [74] Hooman JAROLLAHI, Vincent GRIPON, Naoya ONIZAWA et Warren J GROSS. "Algorithm and architecture for a low-power content-addressable memory based on sparse clustered networks". In : *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 23.4 (2015), p. 642–653.
- [75] Hooman JAROLLAHI, Naoya ONIZAWA, Vincent GRIPON, Takahiro HANYU et Warren J GROSS. "Algorithm and architecture for a multiple-field context-driven search engine using fully-parallel clustered associative memories". In : *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE. 2014, p. 1–6.
- [76] Hooman JAROLLAHI, Naoya ONIZAWA, Vincent GRIPON et Warren J GROSS. "Architecture and implementation of an associative memory using sparse clustered networks". In : *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. IEEE. 2012, p. 2901–2904.

- [77] Yangqing JIA, Evan SHELHAMER, Jeff DONAHUE, Sergey KARAYEV, Jonathan LONG, Ross GIRSHICK, Sergio GUADARRAMA et Trevor DARRELL. "Caffe : Convolutional architecture for fast feature embedding". In : *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, p. 675–678.
- [78] Xiaoran JIANG, Vincent GRIPON et Claude BERROU. "Learning long sequences in binary neural networks". In : *International Conference on Advanced Cognitive Technologies and Applications, Nice, France*. 2012, p. 165–170.
- [79] William B JOHNSON et Joram LINDENSTRAUSS. "Extensions of Lipschitz mappings into a Hilbert space". In : *Contemporary mathematics* 26.189-206 (1984), p. 1.
- [80] A JORDAN. "On discriminative vs. generative classifiers : A comparison of logistic regression and naive bayes". In : *Advances in neural information processing systems* 14 (2002), p. 841.
- [81] MJ KAHANA, MW HOWARD, SM POLYN et HL ROEDIGER. *Learning and memory : a comprehensive reference*. 2008.
- [82] Pentti KANERVA. *Sparse distributed memory*. MIT press, 1988.
- [83] Michael Peter KENNEDY et Leon O CHUA. "Neural networks for nonlinear programming". In : *Circuits and Systems, IEEE Transactions on* 35.5 (1988), p. 554–562.
- [84] Andreas KNOBLAUCH. "Optimal synaptic learning in non-linear associative memory". In : *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE. 2010, p. 1–7.
- [85] Teuvo KOHONEN. "Correlation matrix memories". In : *Computers, IEEE Transactions on* 100.4 (1972), p. 353–359.
- [86] Teuvo KOHONEN. *Self-organization and associative memory*. T. 8. Springer Science & Business Media, 2012.
- [87] Bart KOSKO. "Adaptive bidirectional associative memories". In : *Applied optics* 26.23 (1987), p. 4947–4960.
- [88] Bart KOSKO. "Bidirectional associative memories". In : *Systems, Man and Cybernetics, IEEE Transactions on* 18.1 (1988), p. 49–60.
- [89] Kenneth KREUTZ-DELGADO, Joseph F MURRAY, Bhaskar D RAO, Kjersti ENGAN, Tai Sing LEE et Terrence J SEJNOWSKI. "Dictionary learning algorithms for sparse representation". In : *Neural computation* 15.2 (2003), p. 349–396.
- [90] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON. "Imagenet classification with deep convolutional neural networks". In : *Advances in neural information processing systems*. 2012, p. 1097–1105.

- [91] Stephen LARROQUE, Ehsan Sedgh GOOYA, Vincent GRIPON et Dominique PASTOR. "Using Tags to Improve Diversity of Sparse Associative Memories". In : *Proceedings of Cognitive* (2015).
- [92] Yann LECUN et Yoshua BENGIO. "Convolutional networks for images, speech, and time series". In : *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.
- [93] Yann LECUN, Yoshua BENGIO et Geoffrey HINTON. "Deep learning". In : *Nature* 521.7553 (2015), p. 436–444.
- [94] Yann LECUN, Léon BOTTOU, Yoshua BENGIO et Patrick HAFFNER. "Gradient-based learning applied to document recognition". In : *Proceedings of the IEEE* 86.11 (1998), p. 2278–2324.
- [95] HC LONGUET-HIGGINS, David J WILLSHAW et OP BUNEMAN. "Theories of associative recall". In : *Quarterly reviews of biophysics* 3.02 (1970), p. 223–244.
- [96] Weizhi LU, Weiyu LI, Kidiyo KPALMA et Joseph RONSIN. "Near-optimal binary compressed sensing matrix". In : *arXiv preprint arXiv :1304.4071* (2013).
- [97] Matteo MARIANTONI, H WANG, T YAMAMOTO, M NEELEY, Radoslaw C BIALCZAK, Y CHEN, M LENANDER, Erik LUCERO, AD O'CONNELL, D SANK et al. "Implementing the quantum von Neumann architecture with superconducting circuits". In : *Science* 334.6052 (2011), p. 61–65.
- [98] Arya MAZUMDAR et Ankit Singh RAWAT. "Associative Memory via a Sparse Recovery Model". In : *Advances in Neural Information Processing Systems*. 2015, p. 2683–2691.
- [99] Warren S MCCULLOCH et Walter PITTS. "A logical calculus of the ideas immanent in nervous activity". In : *The bulletin of mathematical biophysics* 5.4 (1943), p. 115–133.
- [100] Robert J MCELIECE, Edward C POSNER, Eugene R RODEMICH et Santosh S VENKATESH. "The capacity of the Hopfield associative memory". In : *Information Theory, IEEE Transactions on* 33.4 (1987), p. 461–482.
- [101] Dean C MUMME. *Storage capacity of the linear associator : beginnings of a theory of computational memory*. Rapp. tech. DTIC Document, 1988.
- [102] Vinod NAIR et Geoffrey E HINTON. "Rectified linear units improve restricted boltzmann machines". In : *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, p. 807–814.
- [103] Kaoru NAKANO. "Associatron-a model of associative memory". In : *Systems, Man and Cybernetics, IEEE Transactions on* 3 (1972), p. 380–388.
- [104] Iku NEMOTO et Makoto KUBONO. "Complex associative memory". In : *Neural Networks* 9.2 (1996), p. 253–261.

- [105] Andrew NG. "Sparse autoencoder". In : *CS294A Lecture notes 72* (2011), p. 1–19.
- [106] Bruno A OLSHAUSEN et al. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images". In : *Nature* 381.6583 (1996), p. 607–609.
- [107] G PALM. "Memory capacities of local rules for synaptic modification". In : *Concepts in Neuroscience 2.1* (1991), p. 97–128.
- [108] G PALM. "On the asymptotic information storage capacity of neural networks". In : *Neural computers*. Springer, 1989, p. 271–280.
- [109] Günther PALM. "Neural associative memories and sparse coding". In : *Neural Networks 37* (2013), p. 165–171.
- [110] Günther PALM. "On associative memory". In : *Biological cybernetics* 36.1 (1980), p. 19–31.
- [111] Seymour A PAPERT. *The summer vision project*. 1966.
- [112] Frank ROSENBLATT. "The perceptron : a probabilistic model for information storage and organization in the brain." In : *Psychological review* 65.6 (1958), p. 386.
- [113] David E RUMELHART, Geoffrey E HINTON et Ronald J WILLIAMS. "Learning representations by back-propagating errors". In : *Cognitive modeling* 5.3 (1988), p. 1.
- [114] Ruslan SALAKHUTDINOV et Geoffrey E HINTON. "Deep Boltzmann Machines." In : *AISTATS*. T. 1. 2009, p. 3.
- [115] Jürgen SCHMIDHUBER. "Deep learning in neural networks : An overview". In : *Neural networks* 61 (2015), p. 85–117.
- [116] Pierre SERMANET, Koray KAVUKCUOGLU, Soumith CHINTALA et Yann LECUN. "Pedestrian detection with unsupervised multi-stage feature learning". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, p. 3626–3633.
- [117] Thomas SERRE, Aude OLIVA et Tomaso POGGIO. "A feedforward architecture accounts for rapid categorization". In : *Proceedings of the National Academy of Sciences* 104.15 (2007), p. 6424–6429.
- [118] Thomas SERRE, Minjoon KOUH, Charles CADIEU, Ulf KNOBLICH, Gabriel KREIMAN et Tomaso POGGIO. *A theory of object recognition : computations and circuits in the feedforward path of the ventral stream in primate visual cortex*. Rapp. tech. DTIC Document, 2005.
- [119] Claude Elwood SHANNON. "A mathematical theory of communication". In : *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001), p. 3–55.
- [120] David SILVER, Aja HUANG, Chris J MADDISON, Arthur GUEZ, Laurent SIFRE, George VAN DEN DRIESSCHE, Julian SCHRITTWIESER, Ioannis ANTONOGLU, Veda PANNERSHELVAM, Marc LANCTOT et al. "Mastering the game of Go with deep neural networks and tree search". In : *Nature* 529.7587 (2016), p. 484–489.

- [121] Friedrich T SOMMER et Günther PALM. "Improved bidirectional retrieval of sparse patterns stored by Hebbian learning". In : *Neural Networks* 12.2 (1999), p. 281–297.
- [122] FT SOMMER et P DAYAN. "Bayesian retrieval in associative memories with storage errors". In : *IEEE Trans. Neural Networks* 9.4 (1998), p. 705–713.
- [123] Nitish SRIVASTAVA, Geoffrey E HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER et Ruslan SALAKHUTDINOV. "Dropout : a simple way to prevent neural networks from overfitting." In : *Journal of Machine Learning Research* 15.1 (2014), p. 1929–1958.
- [124] Karl STEINBUCH. "Die lernmatrix". In : *Biological Cybernetics* 1.1 (1961), p. 36–45.
- [125] Yi SUN, Xiaogang WANG et Xiaoou TANG. "Deep convolutional network cascade for facial point detection". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, p. 3476–3483.
- [126] Ilya SUTSKEVER, Oriol VINYALS et Quoc V LE. "Sequence to sequence learning with neural networks". In : *Advances in neural information processing systems*. 2014, p. 3104–3112.
- [127] Yaniv TAIGMAN, Ming YANG, Marc'Aurelio RANZATO et Lior WOLF. "Deepface : Closing the gap to human-level performance in face verification". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, p. 1701–1708.
- [128] Yaakov TSAIG et David L DONOHO. "Extensions of compressed sensing". In : *Signal processing* 86.3 (2006), p. 549–571.
- [129] Régis VAILLANT, Christophe MONROCQ et Yann LE CUN. "Original approach for the localisation of objects in images". In : *IEE Proceedings-Vision, Image and Signal Processing* 141.4 (1994), p. 245–250.
- [130] Vladimir VAPNIK. *The nature of statistical learning theory*. Springer science & business media, 2013.
- [131] Santosh S VEMPALA. *The random projection method*. T. 65. American Mathematical Soc., 2005.
- [132] Pascal VINCENT, Hugo LAROCHELLE, Yoshua BENGIO et Pierre-Antoine MANZAGOL. "Extracting and composing robust features with denoising autoencoders". In : *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, p. 1096–1103.
- [133] Pascal VINCENT, Hugo LAROCHELLE, Isabelle LAJOIE, Yoshua BENGIO et Pierre-Antoine MANZAGOL. "Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion". In : *Journal of Machine Learning Research* 11.Dec (2010), p. 3371–3408.

- [134] Oriol VINYALS, Alexander TOSHEV, Samy BENGIO et Dumitru ERHAN. "Show and tell : A neural image caption generator". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, p. 3156–3164.
- [135] John VON NEUMANN et Ray KURZWEIL. *The computer and the brain*. Yale University Press, 2012.
- [136] Nicholas WATIER et Charles COLLIN. "The effects of distinctiveness on memory and metamemory for face–name associations". In : *Memory* 20.1 (2012), p. 73–88.
- [137] Paul J WERBOS. "Backpropagation through time : what it does and how to do it". In : *Proceedings of the IEEE* 78.10 (1990), p. 1550–1560.
- [138] David WILLSHAW. *Models of distributed associative memory*. The University of Edinburgh, 1971.
- [139] David WILLSHAW et Peter DAYAN. "Optimal plasticity from matrix memories : What goes up must come down". In : *Neural Computation* 2.1 (1990), p. 85–93.
- [140] David J WILLSHAW, O Peter BUNEMAN et Hugh Christopher LONGUET-HIGGINS. "Non-holographic associative memory." In : *Nature* (1969).
- [141] GV WILSON et GS PAWLEY. "On the stability of the travelling salesman problem algorithm of Hopfield and Tank". In : *Biological Cybernetics* 58.1 (1988), p. 63–70.
- [142] Hugues Nono WOUAFO, Cyrille CHAVET et Philippe COUSSY. "Improving Storage of Patterns in Binary Cluster-Based Neural Networks : Clone-based Model and Architecture". In : *International workshop on Neural Coding, co-located with DATE Conference 2015*. 2015.
- [143] Bolei ZHOU, Agata LAPEDRIZA, Jianxiong XIAO, Antonio TORRALBA et Aude OLIVA. "Learning deep features for scene recognition using places database". In : *Advances in neural information processing systems*. 2014, p. 487–495.

Les réseaux de neurones ont connu un vif regain d'intérêt avec le paradigme de l'apprentissage profond ou *deep learning*. Dans le domaine de la vision par ordinateur, ils permettent de surpasser les performances de l'humain sur des tâches bien spécifiques. Cependant, ces réussites s'obtiennent au prix d'importants moyens de calcul à déployer. En effet, même en bénéficiant des dernières avancées des GPU (processeurs embarqués sur cartes graphiques), les besoins en mémoire et en temps de calcul des réseaux profonds restent significatifs. Par conséquent, l'intégration de tels réseaux à des systèmes embarqués (smartphones, puces à faibles ressources) demeure difficile. Alors que les réseaux dits optimisés, de par l'optimisation des paramètres nécessaires pour réaliser un apprentissage, nécessitent de fortes ressources de calcul, nous nous focalisons ici sur des réseaux de neurones dont l'architecture consiste en une mémoire au contenu adressable, appelées mémoires associatives neuronales. Le défi consiste à permettre la réalisation d'opérations traditionnellement obtenues par des calculs en s'appuyant exclusivement sur des mémoires, afin de limiter le besoin en ressources de calcul.

Dans cette thèse, nous étudions une mémoire associative à base de clique, dont le codage neuronal parcimonieux optimise la diversité des données codées dans le réseau. Cette grande diversité permet au réseau à clique d'être plus performant que les autres mémoires associatives dans la récupération des messages stockés en mémoire. Les mémoires associatives sont connues pour leur incapacité à identifier sans ambiguïté les messages qu'elles ont préalablement appris. En effet, en fonction de l'information présente dans le réseau et de son codage, une mémoire peut échouer à retrouver le résultat recherché. Nous nous intéressons à cette problématique et proposons plusieurs contributions afin de réduire les ambiguïtés dans le réseau. Ces réseaux à clique sont en outre incapables de récupérer une information au sein de leurs mémoires si le message à retrouver est inconnu. Nous proposons une réponse à ce problème en introduisant une nouvelle mémoire associative à base de clique qui conserve la capacité correctrice du modèle initial tout en étant capable de hiérarchiser les informations. La hiérarchie s'appuie sur une transformation surjective bidirectionnelle permettant de généraliser une entrée inconnue à l'aide d'une approximation d'informations apprises.

La validation expérimentale des mémoires associatives est le plus souvent réalisée sur des données artificielles de faibles dimensions. Dans le contexte de la vision par ordinateur, nous présentons ici les résultats obtenus avec des jeux de données plus réalistes et représentatifs de la littérature, tels que MNIST, Yale ou CIFAR. Notre modèle, grâce à sa capacité de généralisation non supervisée, atteint des taux de classification allant jusqu'à 70% de succès et ce, en l'espace de moins d'une seconde avec une consommation mémoire minimale. Ces résultats démontrent que l'utilisation des réseaux de neurones est possible sur des supports à ressources limitées.

The neural networks have gained a renewed interest through the deep learning paradigm. In the computer vision domain, they allow to surpass human performances on specific tasks. However, these achievements are obtained through the use of massive computational resources. Indeed, even with the most recent graphical processor units (GPU), the needs in memory and computation time for neural nets stay significant. Therefore, their integration in embedded systems (smartphones, low resource chips) remains impossible. While the so called optimised neural nets, by optimizing the parameters necessary for learning, require massive computational resources, we focus here on neural nets designed as addressable content memories, or neural associative memories. The challenge consists in realising operations, traditionally obtained through computation, exclusively with neural memory in order to limit the need in computational resources.

In this thesis, we study an associative memory based on cliques, whose sparse neural coding optimises the data diversity encoded in the network. This large diversity allows the clique based network to be more efficient in messages retrieval from its memory than other neural associative memories. The associative memories are known for their incapacity to identify without ambiguities the messages stored in a saturated memory. Indeed, depending of the information present in the network and its encoding, a memory can fail to retrieve a desired result. We are interested in tackle this issue and propose several contributions in order to reduce the ambiguities in the cliques based neural network. Besides, these cliques based nets are unable to retrieve an information within their memories if the message is unknown. We propose a solution to this problem through a new associative memory based on cliques which preserves the initial network's corrective ability while being able to hierarchise the information. The hierarchy relies on a surjective and bidirectional transition to generalise an unknown input with an approximation of learnt information.

The associative memories' experimental validation is usually based on low dimension artificial dataset. In the computer vision context, we report here the results obtained with real datasets used in the state-of-the-art, such as MNIST, Yale or CIFAR. Our model, thanks to its unsupervised generalisation ability, is able to achieve up to 70% of accuracy in classification, in less than a second with a minimal memory consumption. These results demonstrate that using neural networks is possible on supports with limited resources.