N° d'ordre : 2011telb0195

Sous le sceau de l'Université européenne de Bretagne

Télécom Bretagne

En habilitation conjointe avec l'Université de Bretagne Occidentale

Ecole Doctorale – Sicma

Networks of Neural Cliques

Thèse de Doctorat

Mention : Sciences et Technologies de l'Information et des Communications

Présentée par Vincent Gripon

Département : Électronique

Laboratoire : Lab-STICC Pôle : CACS

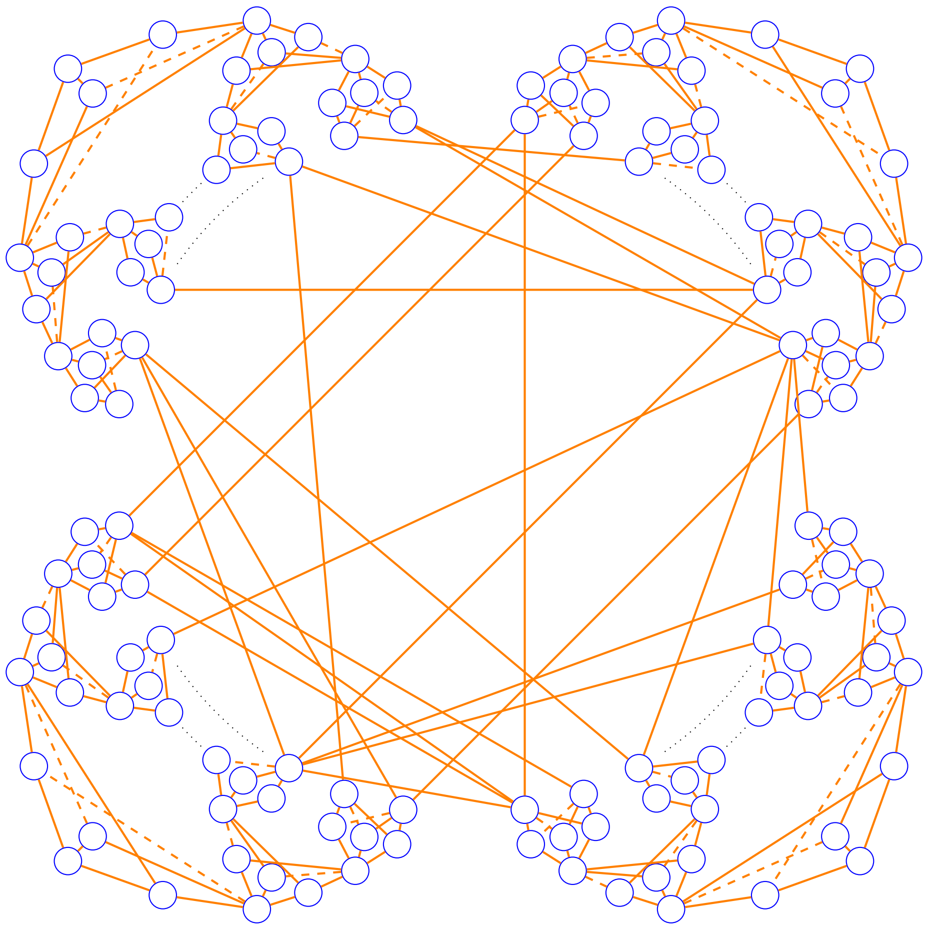
Directeur de thèse : Claude Berrou

Soutenue le 20 juillet 2011

Jury :
M. David Declercq, professeur à l'ENSEA (Rapporteur)
M. Bernard Victorri, directeur de recherche CNRS au Lattice (Rapporteur)
M. Claude Berrou, professeur à Télécom-Bretagne
M. Rodolphe Héliot, chercheur au CEA-LETI
M. Claude Jard, professeur à l'ENS Cachan, antenne de Bretagne
M. Yann LeCun, professeur à l'université de New York
M. Jean-Pierre Nadal, directeur de recherche CNRS à l'ENS Paris
M. Franck Vermet, maître de conférences à l'UBO
M. François Vialatte, maître de conférences à l'ESPCI ParisTech

Version of October 4th, 2011.

Version du 4 octobre 2011.



Résumé

Introduction

Nous proposons et développons un modèle original de mémoires associatives s'appuyant sur des réseaux de neurones codés. Les mémoires associatives sont des dispositifs capables d'apprendre des messages binaires puis de les reproduire à partir de fractions de leurs contenus. L'état de l'art est le réseau proposé par Hopfield, dont la diversité de mémorisation - le nombre de messages qu'il peut mémoriser - est inférieure à $\frac{n}{2\log(n)}$ où n est le nombre de neurones dans le réseau.

Notre travail a consisté à tirer parti des techniques de codage et de décodage correcteur d'erreur, plus précisément celle des codes distribués, afin d'accroître considérablement les performances des mémoires associatives. Pour ce faire, nous avons introduit des codes originaux dont les mots de code sont portés par des cliques neurales. Nous montrons que, combinées à des codes locaux parcimonieux, ces cliques neurales offrent une diversité d'apprentissage qui évolue comme le carré du nombre de neurones.

Les gains observés viennent de l'utilisation de la parcimonie à plusieurs échelles : d'une part les messages appris sont de longueur bien inférieure à n, d'autre part ils n'utilisent qu'une partie du matériel disponible, que ce soit au niveau des neurones ou de leurs connexions. L'apprentissage est donc localisé, au contraire des réseaux de Hopfield. De plus, ces mémoires bénéficient d'une efficacité - rapport du nombre de bits appris au nombre de bits utilisés - presque maximale. Elles se présentent donc comme une alternative intéressante aux mémoires indexées classiques.

Au delà de l'aspect quantitatif, le modèle que nous proposons offre une plausibilité biologique fortement accrue par rapport au modèle de Hopfield. Les concepts de cliques neurales, de *winner take all*, ou encore de synchronisation temporelle que ce modèle exploite rejoignent les observations récentes rapportées par la littérature neurobiologique. Par ailleurs, elles pourraient ouvrir la voie à la conception de machines cognitives capables de croiser des informations pour en produire de nouvelles car les cliques neurales sont recouvrantes, par leurs sommets ou par leurs arêtes.

Réseaux de neurones biologiques et artificiels

Les réseaux de neurones biologiques peuvent être séparés en deux grandes familles. La première se définit par des réseaux *feed forward*, lesquels se prêtent bien à la modélisation des comportements réflexes et aux couches réalisant l'interface entre le corps et le cerveau, le cortex visuel par exemple. La seconde, qui nous intéressera dans ce document, est fortement récurrente et permet la modélisation de fonctionnements à un plus haut niveau informationnel comme la mémoire et le raisonnement.

Afin de modéliser ces réseaux complexes, les réseaux de neurones artificiels utilisent une abstraction adaptée au formalisme scientifique. Ces réseaux peuvent alors à nouveau se séparer en deux grandes familles. La première représente ceux où le temps est crucial et intervient dans des équations différentielles par exemple. La seconde ignore la dimension temporelle et se rapproche du formalisme des automates cellulaires. Nous montrons dans ce document que ce dernier modèle, très simple, suffit pour obtenir de hautes performances de mémorisation sur des réseaux biologiquement plausibles.

Nous comparons notre modèle à l'état de l'art qu'est le réseau proposé par J. Hopfield en 1982. Ce dernier repose sur un graphe de n neurones entièrement interconnectés à l'exception des boucles. Ce graphe est pondéré et symétrique, ce qui signifie qu'il peut être entièrement spécifié à partir de ses $\binom{n}{2}$ connexions.

Un tel réseau peut apprendre puis remémorer des messages binaires de longueur n. On dira qu'un message est appris s'il est possible de le retrouver à partir d'une fraction de son contenu. L'apprentissage se fait en utilisant le poids des connexions. Le nombre maximal de messages qu'un réseau de Hopfield peut apprendre, qu'on appelle sa diversité, est $M = \frac{n}{2log(n)}$.

Cette limite est particulièrement restrictive. En particulier, un réseau de Hopfield est incapable d'apprendre un grand nombre de messages courts, un dictionnaire par exemple. L'apprentissage d'un aussi grand nombre de messages dans un tel réseau nécessite donc qu'ils soient de grande longueur et que le réseau contienne un grand nombre de neurones (plus que de messages).

Pour estimer plus en avant ses performances, il convient d'introduire un second paramètre, que nous appelons capacité. Ce paramètre représente la quantité maximale d'informations binaires qui peuvent être apprises par le réseau. Dans le cas du réseau de Hopfield, où les messages appris sont de longueur n, cette capacité est $C = \frac{n^2}{2log(n)}$.

Il est à noter que la diversité, au contraire de la capacité, ne tient pas compte de la longueur des messages appris. D'un point de vue informationnel, c'est donc la capacité qui représente la limite intéressante du réseau. En revanche, d'un point de vue cognitif, une meilleure diversité est toujours préférable, car elle augmente le nombre de combinaisons et donc les analogies possibles entre messages.

La capacité est à comparer à la quantité d'informations binaires utilisées pour stocker le réseau en mémoire. Cette dernière s'obtient en multipliant le nombre de connexions par le nombre de bits nécessaires pour en quantifier les poids, soit $Q = \binom{n}{2} log_2(M+1)$.

Le rapport $E = \frac{C}{Q}$, appelé efficacité du réseau, représente la quantité de bits des messages appris portés en moyenne par chaque bit représentant le réseau. Dans le cas du réseau de Hopfield, on a $lim_{n \to \infty} E = 0$. Ainsi plus le réseau de Hopfield est grand, moins il est efficace.

Il est à noter que l'efficacité d'un réseau de neurones utilisé en mémoire associative peut dépasser la valeur 1, parce que les messages appris ne sont pas ordonnés.

Réseaux de neurones récurrents

Lors de l'apprentissage, le réseau de Hopfield modifie les poids de l'intégralité de ses connexions, affectant ainsi tout ce qui a déjà été appris. C'est cette approche qui est responsable de la dégradation progressive de la précision de l'information portée par chaque connexion. Cette dégradation se répercute ensuite sur les performances du réseau.

Afin de contourner cette limitation, à chaque message appris peut être associé une localisation différente dans le réseau. Ces localisations, éventuellement recouvrantes, peuvent être de différentes natures : cycles, populations, cliques…

Nous avons dans un premier temps considéré les cycles. Les cycles sont potentiellement très nombreux dans un graphe mais néanmoins limités par leur résilience trop faible : la perte d'une unique connexion peut transformer un cycle en un simple chemin de déroulement fini. De plus, des cycles recouvrants sont difficiles à contrôler, car l'activation de l'un mène souvent à l'activation de certains autres. D'un autre côté, considérer des cycles non recouvrants en limiterait le nombre à celui des noeuds dans le graphe.

Nous avons observé que des réseaux aléatoires, générés de sorte à avoir des paramètres proches de ceux du néocortex, font apparaître des populations de neurones réagissant à des stimuli donnés. Ces populations, s'appuyant sur des cycles, sont malheureusement peu diverses et peu nombreuses.

Notre démarche nous a ensuite conduit à considérer les cliques. Les cliques sont des ensembles de noeuds d'un graphe entièrement interconnectés. Contrairement aux cycles, les cliques sont très résilientes puisque le nombre de leurs connexions augmente comme le carré de celui de leurs sommets. En particulier, après la perte de quelques connexions dans une clique, celle-ci maintient une forte redondance au sein de ses connexions restantes.

Nous proposons alors un modèle simple qui permet d'associer des messages à des cliques virtuelles dans des graphes bipartis. Dans ces graphes, une partie des noeuds représente les neurones du réseau, une autre partie les cliques dans ce réseau. Chaque clique virtuelle est ainsi modélisée par un unique noeud dans la seconde partie. Ce noeud est relié à l'ensemble des neurones appartenant à la clique virtuelle qu'il représente. Ces graphes ont des performances encourageantes puisqu'ils permettent de dépasser la limite sous-linéaire de diversité dans les réseaux de Hopfield pour arriver à une diversité linéaire.

Les limites en performances de ce procédé viennent de la difficulté à contrôler les cliques recouvrantes.

Codes correcteurs d'erreurs et réseaux de neurones

Le domaine du codage correcteur d'erreurs a proposé depuis les années 1950 une pléthore de codes de plus en plus performants. Une découverte majeure a été l'invention des turbo codes au début des années 1990, lesquels ont initié un fort intérêt pour les codes distribués.

Un code correcteur d'erreur a pour rôle d'ajouter de la redondance aux messages afin de les protéger durant leur transport à travers des canaux susceptibles de les bruiter. Le décodeur associé pourra alors tenter de retrouver le message transmis malgré les éventuels erreurs ou effacements survenus lors de la transmission.

Un exemple de code correcteur d'erreurs est celui qui contient tous les mots binaires sur $\{0;1\}$ de taille n et qui n'ont qu'un seul bit de valeur 1. Étant donné son faible coût en caractères 1, nous avons baptisé ce code "code économe". Le code économe est relativement peu robuste puisqu'une unique erreur qui insérerait l'un quelconque des symboles d'un mot de code mènerait à une ambiguïté lors de son décodage. En contre partie, ces mots de code sont particulièrement simples à décoder. En effet, si les mots sont bruités par l'addition d'une valeur aléatoire et indépendante sur chacun des bits, le mot de code le plus probablement émis est celui qui contient un 1 là où la valeur reçue était maximale. Cette règle de décodage trouve une parfaite correspondance dans la littérature neurobiologique sous le nom de *winner-take-all*.

Nous introduisons également un code original, gratuit et performant qui existe dans n'importe quel graphe. Ce code est simplement constitué des cliques contenant c noeuds de ce graphe. Il est particulièrement performant. Deux cliques de tailles c différent au minimum de seulement deux noeuds (un noeud ayant été remplacé par un autre), mais en termes de connexions ce nombre devient

$2(c-1)$. Par ailleurs, seules $\lceil \frac{c}{2} \rceil$ connexions suffisent à identifier sans ambiguïté une telle clique, alors qu'elle en contient $\binom{c}{2}$ au total. La comparaison du nombre de cliques de taille c à leur distance minimale permet de conclure que les cliques de taille c forment les mots de codes d'un code intéressant (même pour des applications en télécommunications). De plus, les cliques ne requièrent aucun matériel supplémentaire dans un réseau de neurones : elles sont naturelles. Enfin, les neurobiologistes sont familiers avec la terminologie de cliques neurales, puisque celles-ci ont été observées dans certaines régions du néocortex.

Les autres codes correcteurs d'erreurs peuvent avoir des propriétés analytiques intéressantes, comme par exemple l'orthogonalité de leurs mots de code : c'est le cas du code de Hadamard par exemple. Ces codes peuvent être directement introduits dans les réseaux de Hopfield pour accroître leur diversité d'apprentissage, en leur apportant à la fois le bénéfice de leur orthogonalité et de leur pouvoir de correction. Les messages à apprendre sont alors directement associés à des mots du code considéré, lesquels deviennent le coeur du processus de décodage.

La diversité d'apprentissage peut encore être accrue en combinant ces codes les uns avec les autres à la manière des codes distribués. Cette association nécessite cependant l'ajout d'un autre code plus grand et plus puissant. Les gains en diversité observés sur des réseaux de taille moyenne (environs 1000 neurones) sont d'un facteur de l'ordre de 50.

Ces diverses techniques permettent de considérablement accroître les performances d'apprentissage des réseaux récurrents, mais les ajouts sont techniquement complexes et écartent toute plausibilité biologique.

Parcimonie et réseaux de cliques neurales

Afin de mieux tirer parti des propriétés des codes distribués tout en respectant la contrainte de plausibilité biologique, nous proposons d'utiliser à présent des codes économes et des codes sur cliques. Ces codes, introduits dans le chapitre précédent, sont compatibles avec cette contrainte.

Cette association nous permet de proposer un réseau très performant qui fonctionne de la façon décrite ci-après.

D'abord, les messages binaires à apprendre sont découpés en sous-messages de taille κ. Ces sous-messages sont associés de façon unique à des mots d'un code économe plus long de taille $l = 2^{\kappa}$. Cette transformation étant bijective, il est équivalent d'apprendre l'association des sous-messages de départ ou d'apprendre celle des mots des codes économes correspondants.

Chacun des mots de codes économes ainsi obtenus est projeté sur une partie différente du réseau, appelée grappe ou cluster, chacune de taille l (une grappe est associée aux premiers sous-messages, une seconde aux seconds…). Cette projection n'active qu'un unique neurone dans chaque grappe, celui correspondant au bit à valeur 1. Ces différents neurones sélectionnés, que nous appelons fanaux parce que seul l'un d'entre eux est allumé dans la pénombre de sa grappe, sont ensuite entièrement reliés deux à deux, ce qui inscrit une clique dans le réseau. Ces connexions ne pouvant qu'être entre parties distinctes, le réseau est multiparti. Il est également entièrement binaire, ce qui signifie qu'une connexion ne sera pas renforcée si elle est utilisée plusieurs fois. Cette différence avec le réseau de Hopfield, très importante vis-à-vis de la plausibilité biologique, vient du fait que c'est la présence de connexions qui code l'information plutôt que leurs poids, le graphe n'étant pas complet.

La remémoration d'un message appris à partir d'une fraction de son contenu s'effectue ensuite en exploitant les deux codes. D'abord, les sous-messages ayant une information, même partielle, sont projetés sur leurs fanaux correspondants. Ensuite, les activations de ces neurones sont transmises via les connexions du réseau, en particulier celles de la clique correspondant au message à retrouver. D'autres connexions peuvent être utilisées lors de cet échange de signaux, elles agissent alors comme du bruit. Les neurones se comportent ensuite comme des intégrateurs qui additionnent les signaux entrants. Le second décodage, correspondant aux codes économes, peut alors débuter : dans chaque grappe, seul le (ou les) fanal le plus actif est sélectionné.

Il se peut que plusieurs fanaux, à l'intérieur d'une même grappe, atteignent le score maximal, menant alors à une ambiguïté qui ne peut être levée que par des itérations supplémentaires des deux décodages : global par les cliques puis local par les codes économes.

Nous montrons que ces réseaux compartimentés en c grappes de tailles l peuvent apprendre un nombre considérable de messages, proportionnel au carré du nombre de neurones $n = cl$, si c est constant. Leurs capacités sont très voisines des quantités d'informations binaires nécessaires pour stocker le réseau en mémoire, ce qui conduit à des efficacités proches de 1, bien supérieures à l'efficacité tendant vers zéro des réseaux de Hopfield.

Ces bonnes performances s'accompagnent d'une meilleure robustesse et flexibilité des réseaux. En particulier, les connexions sont binaires et donc bien plus résistantes au bruit et la taille des messages appris n'est plus donnée par la taille du réseau.

Nous proposons ensuite plusieurs développements, dont la considération des sources corrélées et de messages de longueurs variables. Dans le premier cas, nous montrons que l'addition de redondance aléatoire aux messages à apprendre permet de combattre l'effet néfaste de la corrélation sur les performances du réseau. Dans le second cas, nous montrons que l'exploitation de la dimension temporelle, qui n'avait pas été utilisée jusqu'alors, permet de sélectionner lors du décodage les parties qui doivent être utilisées. Cette sélection repose sur la détection de coïncidences temporelles.

En dehors de leurs performances, ces réseaux sont très plausibles du point de vue biologique. Les cliques neurales et le décodage par *winner-take-all* trouvent des échos dans la littérature neuroanatomique. Le découpage en grappes est analogue à la structure en colonnes du néocortex. Les fanaux sont spécifiques des valeurs de certains sous-messages, au même titre que des neurones du cortex visuel peuvent être spécifiques à la détection d'une caractéristique particulière (un angle précis par exemple). Enfin, le réseau converge en pratique avec un nombre très faible d'itérations vers un point fixe, au contraire des réseaux de Hopfield.

Conclusion, ouvertures

Les réseaux que nous avons introduits dans ce travail de thèse ont permis de considérablement augmenter les performances des mémoires associatives. Ces gains s'observent à la fois en diversité et en capacité.

Ces résultats ont pu être obtenus grâce au principe de parcimonie et à la puissance des codes correcteurs d'erreurs distribués comme ceux de type *LDPC*, par exemple.

Un résultat important de ce travail est l'étonnante constatation qu'aucun compromis n'a été nécessaire entre plausibilité biologique et performances.

Il reste que ce travail n'est qu'un point de départ qui ouvre une immense potentialité de développements dans de nombreux domaines, dont certains sont introduits ci-dessous.

D'abord, des questions se posent sur le transport de ce modèle vers le monde de la biologie. Notamment, l'introduction des comportements temporels de neurones à *spikes* s'impose alors comme une étape incontournable. Plus spécifiquement, ce réseau pourrait être suffisamment riche pour modéliser des pathologies liées à la mémoire et aux problèmes de synchronisation.

Dans les domaines de l'informatique et de l'électronique s'ouvrent des perspectives diverses, allant de la conception de circuits dédiés à l'exploitation dans la conception d'algorithmes innovants. Il est certain que ces mémoires requièrent des paradigmes de programmation spécifiques, bien différents de ceux qui sont habituellement utilisés.

Enfin ces mémoires ouvrent la voie à des conceptions de machines cognitives originales. Leur utilisation nécessitera probablement l'utilisation d'ontologies adaptées. Un travail important est à faire sur les notions de pertinence et d'attention, qui sont indispensables pour favoriser l'émergence de “consciences” virtuelles. Ces réseaux de neurones ont pour particularité de représenter les messages sur des cliques recouvrantes, par leurs noeuds et par leurs arêtes, et ouvrent donc la voie à de nombreuses pistes sur leur capacité à combiner ces messages pour en créer de nouveaux, que l'on espère pertinents.

Abstract

We propose and develop an original model of associative memories relying on coded neural networks. Associative memories are devices able to learn messages then to retrieve them from part of their contents. The state-of-the-art model is the Hopfield Neural Network, whose learning diversity - the number of messages it can store - is lower than $\frac{n}{2\log(n)}$ where n is the number of neurons in the network.

Our work consists of using error correcting coding and decoding techniques, more precisely distributed codes, to considerably increase the performance of associative memories. To achieve this, we introduce original codes whose codewords rely on neural cliques. We show that, combined with sparse local codes, these neural cliques offer a learning diversity which grows quadratically with the number of neurons.

The observed gains come from the use of sparsity at several levels: learned messages length is much shorter than n, and they only use part of the available material both in terms of neurons and connections. The learning process is therefore local, contrary to the Hopfield model. Moreover, these memories offer an efficiency - ratio of the amount of bits learned to the amount of bits used - nearly optimal. Therefore they appear to be a very interesting alternative to classical indexed memories.

Beside the performance aspects, the proposed model offer much greater biological plausibility than the Hopfield one. Indeed, the concepts of neural cliques, winner-take-all, or even temporal synchronization that we introduce into our networks match recent observations found in the neurobiological literature. Moreover, since neural cliques are intertwined by their vertices and/or their connections, the proposed model offers new perspectives for the design of cognitive machines able to cross pieces of information in order to produce new ones.

“La Science remplace du visible compliqué par de l’invisible simple.”
“Science replaces complicated visible with simple invisible.“
Jean Perrin,
Nobel laureate in physics.

To my parents

Acknowledgements

First of all, I would like to thank my thesis advisor Claude Berrou. It was really an honour having the opportunity to work with him during these three years and beside his qualities as an eminent scientist he truly inspired me by his humanity, his pedagogy, and his intuition. I learned a lot of things thanks to him during my thesis and most of them are not written in this document.

I could not help but commend my former school, the École Normale Supérieure of Rennes which provided me with a top education that eased the research I have done since. Once again, it is far more than a simple scientific education that I would like to give thanks, it is also a formidable and very rewarding human experience. I would especially like to thank Luc Bougé and Claude Jard who head the computer science and telecommunication department of this school and commend their excellent work.

Although he did not directly help me during my thesis, I would also like to thank Olivier Serre, my former (internship) advisor. He has been a mentor to me and helped me gain confidence. He has also been a great source of inspiration.

My thanks then go to Michel Jezequel, the director of the Electronic Department of Télécom Bretagne, for his wise advices and his help in a lot of projects. He did much to put me at ease throughout my thesis research.

I thank the reviewers - David Declercq and Bernard Victorri -, members of the jury - Claude Berrou, Rodolphe Héliot, Claude Jard, Yann LeCun, Jean-Pierre Nadal, Franck Vermet and François Vialatte - and invited guests of my thesis defense for showing interest in our work. I hope we will come to share a little of the spark that has been animating Claude and I for three years.

I thank all the Ph.D. students with whom I interacted, especially Nicolas Bitouzé and Xiaoran Jiang.

I also thank Professor Warren Gross, of McGill University, for accepting the burden of correcting my English.

I am very grateful for all the talented teachers I had during my education, for the quality of life and the good weather we have in Brittany, and also for all those who supported me at some point during my life.

Finally, I would like to thank my parents for their unwavering support throughout my life. They were always there for me and considerably helped me into developing my interests for a lot of things, including research. It is clear that I would not have gone so far without their constant love and support.

Contents

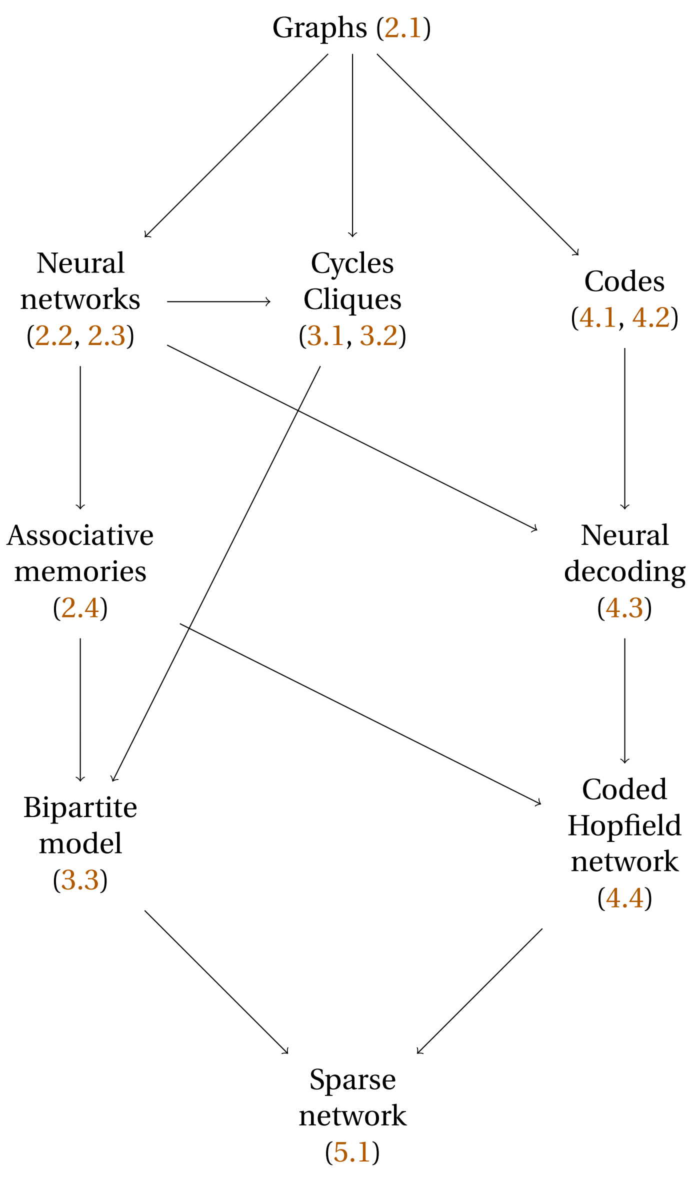

Graphs (2.1)
Neural networks (2.2, 2.3)
Cycles Cliques (3.1, 3.2)
Codes (4.1, 4.2)
Associative memories (2.4)
Neural decoding (4.3)
Bipartite model (3.3)
Coded Hopfield network (4.4)
Sparse network (5.1)

Figure 1: Direct dependencies and sections in the document.

1 Introduction

"I visualize a time when we will be to robots what dogs are to humans" said Claude Shannon, the father of Information Theory. Most people probably do not consider we are close to this time. Maybe they ignore that supercomputers are able now to process up to 10^{15} elementary operations per second. In comparison, a human brain neocortex has typically some 10^{10} neurons with a response time of 10^{-3}s, that is to say approximately the same computing ability. One could argue that human brains are not comparable with Turing machines: if the former can emulate the latter (considering infinite time), too many metaphysical questions arise to consider the reciprocity.

We are probably close to a singularity, as described by futurists like Ray Kurzweil, which shall correspond to the moment when machines will be able to store, retrieve, merge and produce pieces of information by themselves. Added to their effective ability to communicate, for instance through the Internet, this would lead to an incredible increase in overall knowledge. Not only will those machines considerably help any research, but they will also likely be able to lead to the design of higher performance machines and so on.

Yet it is still unclear how these machines could be developed. Although computing abilities are following an exponential increase, we lack the methods to efficiently manipulate cognitive[1] information. Actually, there is just one known device able to do it: the brain. Obviously, machine designers are not the only community of scientists interested in studying the mechanics of the brain. Neurologists, psychologists, philosophers, and many more are willing to understand its functioning. However, and despite the astronomic quantity of intelligence involved, some basic questions have not yet found a convincing answer. Among them, the mental principles of information storage are still unknown. Yet there have been some proposals, associative memories being one of them.

[1]Cognition, according to Wikipedia, "refers to information-processing abilities of humans".

Associative memories are devices that mimic the brain memory in some aspects: they can learn messages and retrieve them in presence of errors or erasures. Of particular interest are such devices targeting biological plausibility as they offer an opportunity to decrypt the principles of information storage in the brain.

Evolution eventually gave man a complex neural network, structured into micro, macro and hyper-columns whose roles are partially known. This increasing knowledge of the brain's organization allows neurologists to draw constantly more accurate semantic maps of it. Of course, the interest of such an approach is unquestionable as far as medical issues are concerned. Nonetheless, it seems less obvious with regards to the question of information in the brain. There are many reasons justifying this remark. One among them comes from cellular automaton theory, which describes models with very simple dynamic rules that can result in chaos. Consequently, the observation of such automaton evolution cannot lead to the understanding of its rules. The brain is, of course, a formidable machine with very complex chemical rules. It seems therefore difficult to imagine that the observation of its architecture could be sufficient to deduce its functioning principles.

It is a surprising fact that there exist only a few connections between information theory and cognitive sciences. All the more since the term "artificial intelligence" originally proposed by McCarthy was largely expanded thanks to the conference of Dartmouth in 1956 where one of the main speakers was Claude Shannon. Actually, some papers cross the domains, as for instance some works using error correcting codes to increase perception learning process [1] or on the other hand decoding with the help of artificial neural networks [2]. These contributions are nevertheless limited to the search for optimization of already existing techniques.

This observation summarizes the surprising context in which the work described in this document began: principles of information storage in the brain are an open issue which is not considered by information theorists. Yet Claude Berrou[2] has been presenting in the last few years a large list of similarities between the functioning of modern error correcting decoders and our knowledge of the brain: uniqueness of thoughts and fixed point decoding, resilience and noise resistance, macro-columns versus small local codes, etc... These observations, illustrated in Figure 1.1, are the guiding principles of this thesis.

[2]Claude Berrou is mainly known as the inventor of turbo codes, a family of error correcting codes that achieves near optimal performance and that dramatically improved the previous state-of-the-art in the early 90's.

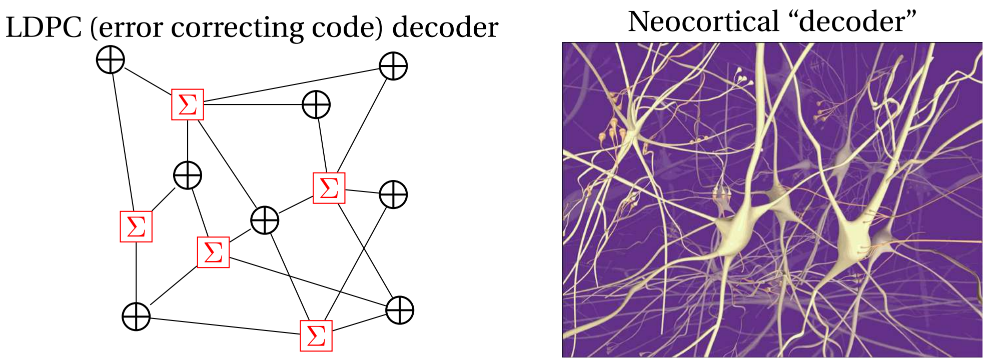


Figure 1.1: Error correcting decoding versus neocortex decoding. We can draw a parallel between the adder nodes of the LDPC graph and the neurons. But there is no direct correspondence for the parity nodes.

As a matter of fact, error correcting decoders are very similar to associative memories: they find the most likely message associated with a given input. Yet similarities have their limits, as on the one hand decoders treat intrinsically dependent messages (generally linearly dependent) and associative memories consider arbitrary ones.

The results presented in this document are obtained using two distinct measures of performance: the first one is the capacity, that is to say the total amount of information a device can learn, in bits[3]. The second one, the diversity, is the number of messages the same device can learn. If the former has been significantly increased by our work, the latter has dramatically grown. As far as cognitive issues are concerned, the diversity of a network seems to be the most important parameter. Indeed, it is better to be able to learn a lot of short messages than to learn a small amount of long messages as it is the ability to confront many pieces of information which is the very foundation of human thinking.

Given this fact, it is easily understandable that the most important word in this document will be sparsity. Sparsity will be introduced at multiple levels to allow us to reach unprecedented performance. Actually, the very principle of learning is based on sparsity. Let us imagine, for instance, that one want a neural network to learn the word "brain". Our idea is that this word will be projected onto a geometric figure. Such a process is depicted in Figure 1.2. Actually, the strategy is simple: the word "brain" is only made of five characters and is therefore close to many other words such as "train". Consequently, an erasure on a single character would likely result in an ambiguity. By considering words as connections between characters, the minimum number of distinct connections between the

[3]According to broad estimates, the amount of bytes memorized in a human brain during its all life is around 110^9 (Ralph C. Merkle, Foresight Update, No. 4, Oct. 1988).

letters of “brain” and another five letter word becomes at least eight, when a single character has changed. In other words, the small initial space of words with five characters has been projected to a bigger space: the one of connections between characters. The advantages are many. A first one is that a bigger space allows a larger minimum distance between messages, giving a better separability. Additionally, when two characters differ, a lot of connections also differ: the learning thus adds a spatial diversity. On top of that, the material used to learn is connections, which are quadratically more numerous than characters.

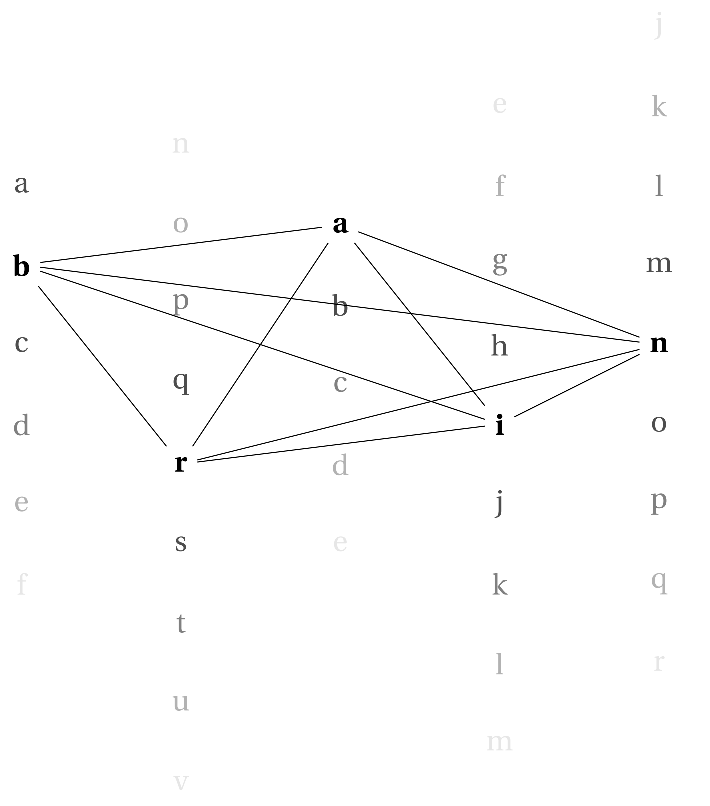


Figure 1.2: A simple strategy to learn the word “brain” with an error resistant scheme.

The objective of the work described in this document was originally to explore whether the human brain can be seen as an error correcting decoder or not. It is why it constantly oscillates between biological plausibility and effectiveness. It is remarkable that eventually both motivations are completely fulfilled without having needed to ignore one to enhance the other.

In order to have a clear presentation, important results are presented in boxes like this one.

If this result is also an original contribution of our work, the box will be presented with double lines.

Actually, we will present a lot of strategies to combine error correcting codes with associative memories in order to improve their performance tenfold. Those converge at the end of the document to a very effective and biologically plausible network (Chapter 5). This network is at the heart of the work presented in this document and all the preceding chapters just aim at introducing it.

In the second chapter, we present the tools that are at the basis of this work: graphs, and then biological and artificial neural networks. The Hopfield model for associative memories is also analysed, as it will serve as the state-of-the-art reference.

The third chapter emphasizes our interest for cycles as support of information in recurrent neural networks. Actually, if those cycles are numerous, they do not necessarily lead to dynamically observable properties. We then consider populations of neurons and finally cliques (fully interconnected subsets of neurons). The latter appear to be the perfect candidate for storing pieces of information. A very simple and effective original associative memory, exploiting cliques, is then analysed.

Chapter 4 introduces the notion of error correcting code through classic examples (repetition code, Hadamard code). Some important contributions of our work are presented in this chapter, such as a result on optimal constant weight codes and a completely original code that will be at the very foundation of the model described in Chapter 5: clique-based codes. Then original neural error correcting decoders are presented working on any code, yet with a large number of neurons. Finally, the Hopfield model is mixed with error correcting codes, providing a very large increase of performance.

Chapter 5 finally introduces the main contribution of this thesis: sparse neural networks. Those networks present unprecedented performance with very interesting biological plausibility. Various improvements are proposed, from the consideration of correlated inputs to the construction of distributed sparse subnetworks to get around performance limitations.

The different issues that are considered in each chapter are listed in the following list.

- **Chapter 2:** *What are graphs? How can one model the neocortex? What are artificial neural networks? What are Hopfield neural networks (HNN) and what are their limitations?*

- **Chapter 3:** *Why are cliques good candidates for the storage of pieces of information in recurrent neural networks? How is it possible to associate pieces of information with cliques? What is the performance of the obtained networks compared to HNN?*
- **Chapter 4:** *What are error correcting codes? Is it possible to design efficient easily decodable error correcting codes on graphs? Are neural networks able to decode such codes? How can we mix efficient codes and associative memory to bypass the HNN limitations?*
- **Chapter 5:** *Why is adding sparsity a very good way to improve performance of associative memories? Can we benefit from sparsity and error correcting codes conjointly? What is the performance of the obtained networks with respect to HNN? How can we improve furthermore performance in these networks?*

2 Graphs, biological and artificial neural networks

Contents

In this chapter, graphs and associative memories are introduced. Those tools will be at the basis of all the results presented in this document. The biological and artificial neural networks are also discussed.

2.1 Graphs

Graphs were originally introduced by Euler in 1741 [3]. In this famous paper, he proved one could not walk passing by every bridge of the city of Königsberg once and only once. Such a path that contains every edge of a graph has since been called Eulerian.

The formalism he introduced has since been largely studied, especially these past decades. So much so that several domains such as computer science use it at every level, from the design of circuits to the research of winning strategies in abstract games (for example, but not by chance [4]).

Yet the model is simple: a graph is a set of vertices - or nodes - connected through edges - or connections. The latter can be weighted and/or directed, for instance if they represent a distance map between cities in Brittany. Actually, graphs can be arbitrary complicated to fit a specific domain of application.

Formally, a graph is a couple $< \mathcal{V}, \delta >$, where:

- $\mathcal{V}$ is the set of vertices,
- $\delta \subset \mathcal{V} \times \mathcal{V}$ represents the set of edges.

This definition is the largest possible, in the sense that it covers any possible elaboration of the model, and is equivalent to the definition of a mathematical relation. An example of such a directed graph is that associated with the relation $\leq$ on integers, containing a non finite number of vertices and of edges.

If the graph is weighted, for instance over $\mathbb{R}$, δ becomes a function $w : \mathcal{V}^2 \rightarrow \mathbb{R}$ which associates a weight with any edge.[1]

Some particular graphs are noteable, for instance a graph that contains no connections - called an empty graph - or all the possible edges - called a complete graph. For instance, the graph of euclidean distances between cities of Brittany is a complete graph. Sometimes, a graph with all edges except for those between the vertices and themselves - called loops - are also called complete graphs; this convention would hold only if loops are nonsense.

Also, some graphs have the property that their set of vertices can be split into a partition such that every subset contains no edges; these are called multipartite. In this case, it is usual to specify (minimum) number of subsets when possible: one could consider bipartite or tripartite graphs for instance. Figure 2.1 depicts a bipartite graph with a typical representation: vertices are represented by circles or squares, depending on their belonging to the first or to the second subset. As it is a bipartite graph, there is no connection between two circles or two squares. Note that by definition a bipartite graph cannot contain any loops. The graph that models the associations between the cities of Brittany and their belonging to any of the historical five departments is another example of a bipartite graph.

It is often useful to colour the graph, which means to provide it with a function $C : \mathcal{V} \rightarrow \mathcal{C}$ where C can be any set, which associates a colour with each vertex. The colour can represent the measure of various concepts. For example, the previous bipartite graph representing cities of Brittany and their belonging to

[1] In all cases considered in the sequel the absence of a connection is equivalent to the same connection with weight 0, such that weighted graphs are a generalization of graphs.

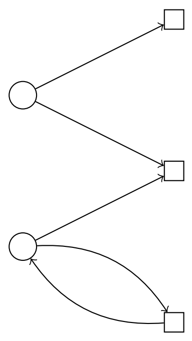


Figure 2.1: Example of a bipartite directed graph.

departments could be coloured by the population of cities. Colours can also be introduced on connections, the weight being an example.

Many other aspects of graphs will be discussed later in this document, and will be introduced when needed.

2.2 Biological neural networks

Neurons are specific types of cells found mainly, but not only, in the brain of animals. They can be sensors or actuators, as well as devoted to memory functionalities. Their connections materialize a network. Those are called synapses and axons; they allow the transport of signals from one neuron to another using spikes.

Impulses coming from neurons will either favour or inhibit the excitation of other ones. If those excitations surpass some threshold, these are likely to fire. When a neuron fires, it releases a fresh impulse through the outgoing axon. This impulse will then supply synapses connected to other neurons.

Chemically speaking, this dynamic is explained by an electrical potential that exists in neurons. A chemical reaction only occurs if this potential is sufficient. If it does, the potential is reset to a lower value while the impulse is transmitted. The neuron is not likely to transmit again soon after it fires, providing the neurons with a relaxation time.

It seems that there exist a lot of types of neurons in the human brain [5], having possibly different behaviours. A similarity in their functioning is that they have the ability to aggregate their different inputs and to somehow combine them into a global information signal.

The ability a man has to store and combine information is given by the properties of his neocortex[2]. The human neocortex is a hull composed of six layers which cover the human brain. It has a particular organization of its neurons in columns[3]. A typical human brain contains about half a million of micro-columns, grouped into macro-columns, themselves grouped into hyper-columns. Micro-columns contain about a hundred neurons each. The neocortex is known to be devoted to some functions as language and conscious thoughts.

With this specific cognitive architecture comes a particular connection distribution. Those are very present locally, inside micro-columns and with close ones. Distant connections are sparse. This distribution has been the source of inspiration for a part of the scientific community, modelling it as a small world network [6]. This consideration is however controversial.

Our idea is that it is not necessary to understand the extremely rich and complex chemistry of the brain to propose biologically plausible models for the information storage principles. Actually, we began our work considering the neurons as fundamentally “nodes in a graph”. This is not a bad idea according to the medicine Nobel price Eric Kandel who wrote “Learning and memory cellular mechanisms do not reside in the specific properties of the neuron itself but in the incident and outgoing connections it establishes with the other cells of the neural circuit it belongs to”[4]. We discovered recently that the original network, presented in Chapter 5, matches perfectly with how the functioning of the neocortex is described by some neurologists. This functioning is described in [7] where memory is linked to the notion of neural cliques, in [8] where sparse coding and time synchronization are used to represent pieces of information and in [9] where the neural network is split into clusters where only the neuron with the maximal activity has the ability to fire. This latter rule, introduced in the following subsection, is called winner-take-all.

2.2.1 The winner-take-all rule

A very important and recurrent rule found in the neurobiologic literature is the winner-take-all ([10,11]). It corresponds to a selection of the neuron (or the small set of neurons) that achieves maximum activity in a given region of the neural network.

[2]The name neocortex comes from the fact it is considered to be the latest contribution of evolution to the brain.

[3]They are called columns as they are orthogonal to the layers of the neocortex.

[4]“Les mécanismes cellulaires de l’apprentissage et de la mémoire ne résident pas dans les propriétés spécifiques du neurone lui-même, mais dans les connexions entrantes ou sortantes qu’il établit avec les autres cellules du circuit nerveux auquel il appartient,” *À la recherche de la mémoire*, Eric Kandel, Nobel prize of Physiology or Medicine, 2006

This rule can be compared to an election in a democratic system. The candidate that achieves the most important score is the only one authorized to express itself, as an echo to the (relative) majority.

Yet the neuroscience community is divided on the question of the implementation of this selection process. Some models consider a construction of this selection using specific architectures of neurons [12]. Others attribute to glial cells a crucial role in this selection process [13].

Figure 2.2 illustrates the functioning of this rule.

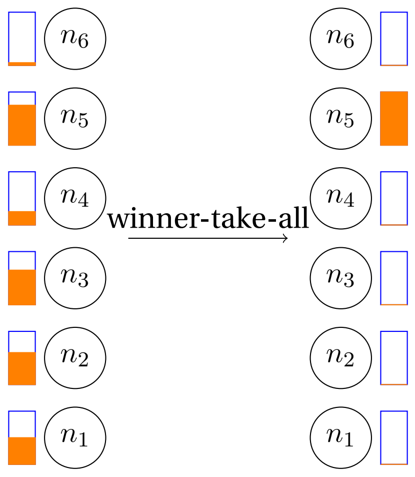


Figure 2.2: Illustration of the winner-take-all rule. The filled partially filled rectangles represent the activity level of the associated neurons.

2.2.2 Sparse coding

It is important to clarify the sparsity concepts that are going to be considered in this document. We introduce sparsity at different levels in the neural networks [14].

The first level is that of information itself. Attentiveness is such that, at a precise time, the brain is focused on only a few primary elements of thoughts (which have been estimated to be seven on average [15]). This means that among the generous amount of information received at this time, only a small part of it will be considered relevant and actually used.

Once in the neocortex, this information will address a restricted part of neurons, most often in specific, specialized regions of the brain. This induces another level of sparsity, which corresponds to the regions used at a specific time.

The winner-take-all rule introduces another level of sparsity, which is more local, and finds a direct correspondence to precisely what biologists call sparse coding in the brain [16]. They present it as the fact that only a few neurons are firing at a specific time.

Finally, a last level of sparsity is given by the network connections. It has been estimated that the number of connections from a neuron in the human brain is about 10^4. Compared to the number of neurons ($\approx$ several 10^{10}), this shows the very low average density of the neocortex graph, which could be represented by a sparse connection matrix.

It is not a surprise that sparsity is present at every level of the information processing in the brain. Actually, sparsity has a direct analogy to energy minimisation which is often a good explanation of the life strategies.

2.3 Artificial neural networks

If biology offers a large amount of literature on biological neural networks, computer science has also proposed its share for artificial ones.

These models can be split into two categories. In the first category, the dynamics of the entire network is given by that of neurons, taken independently. This means that the overall system can be seen as a cellular automaton. In the second category, the dynamics of all neurons is set at once through a differential (or iterative) system. The second model is obviously richer as it contains the first one, the reciprocal predicate being false. On the other hand, considering high order partial differential equations can result in intractability.

The main difference when considering a neural network rather than a graph is that a time-based evolution will be considered. In the case of neural networks, some values are time-dependent, such as the input/output of neurons. In the sequel, this time (or iteration) will be denoted as an exponent t on any notation.

We point out that the notion of time which is introduced here is orthogonal to that which may be contained in the processed information.

First category:

In the first category, neurons receive different inputs and produce an associated output. One can restrain this general model to the one of Figure 2.3, considering that the aggregation process is comparable to a sum of the different inputs.

Most often, the f function possesses a threshold σ such that when surpassed the neuron fires (in the case of this static model, saying that the neuron fires means that it becomes active). If we postulate that the output value is set to 0 when the neuron is inactive, and 1 otherwise, which will be considered true in

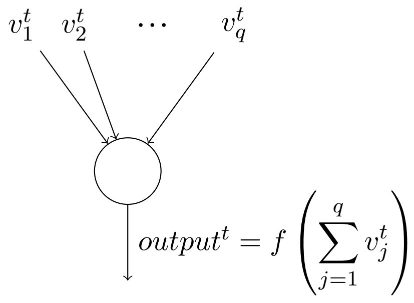


Figure 2.3: Generic neural model used in this document.

the sequel, then the function f can be written as:

$$f(v) = \begin{cases} 1 & \text{if } v \geq \sigma \\ 0 & \text{otherwise} \end{cases}$$

Actually, this writing captures a lot of neuron models frequently encountered in the literature and will be adequate for all those presented in this document.

Neurons are then connected through an edged-coloured graph. An edge-coloured graph is a tuple $< \mathcal{V}, w, \mathcal{C} >$ where:

- $\mathcal{V}$ is the set of vertices (here: neurons),
- $w : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is the edge function,
- $\mathcal{C} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^s$ is a colour function on edges - and not on nodes.

The colour function associates some properties with each edge such as time propagation or frequency filtering considerations. Except for a specific network presented in the last chapter (5.3.2.2) of this document, the considered networks will never need this colouring function.

Second category:

The other category of neural networks considers a higher level of neuron dynamics. An example in this category is the Kohonen model [17]. This model is used mainly in classification and is thus provided with two phases: learning and retrieving.

In the Kohonen model, neurons are connected according to an underlying grid, in such a way that they describe a map. The functioning has two phases: first, during the learning phase, weights are updated such that some regions of the grid are made more responsive to the input that is currently presented to the

Model	Learning	Retrieving
Perceptrons [18]	Back-propagation [19]	Feedforward
Hopfield neural networks (and the original model we propose in Chapter 5)	Feedforward	Iterative
Boltzmann machines [20]	Iterative	Iterative

Table 2.1: Table of three popular neural networks models using different algorithms for the learning and the retrieving phases. The type of the used algorithm for each phase is indicated.

network. Then, during the retrieving phase, an input is presented to the network and only the neuron that achieves the maximum activity is activated. This behaviour corresponds to the winner-take-all rule previously introduced. The learning also echoes to biological observations in the brain: namely Hebb's rule[5].

Actually, in classification and associative memories, many artificial neural networks models use different algorithms for the learning and retrieving phase. Table 2.1 gives some famous examples.

Those two models correspond to two distinct, but yet complementary, visions of the brain. In the original network presented in Chapter 5, both aspects are present.

2.4 Hopfield model

2.4.1 Associative memory state-of-the-art

Among all these models, Hopfield neural networks (HNN) [21] are the state-of-the-art reference we will consider in associative memories. Actually, a lot of work has been done since the introduction of this model but the efficiency (which is introduced below) has not been increased.

An associative memory, or more precisely an autoassociative memory, is a device able to learn then retrieve a set of messages. The learning process is done in perfect knowledge of messages whereas the retrieving one is assured in presence of errors or erasures. More precisely, associative memories that are of interest to the neuroscience community are those using a biologically plausible material; basically a neural network. To ease the comparison of performance, most of the models introduced in this document will consider binary messages. Yet some

[5]Hebb's rule was published in 1949 as “The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other.” Source: Wikipedia

of them, as the sparse neural network presented in Chapter 5, could trivially be adapted for any kind of messages.

If one is not concerned about plausibility, he can design an optimal associative memory. This device would just put every message it has to learn in a memory. Then, during the retrieving, it would compare every of those with the one presented as its input and select one (or the one) that best matches it. Obviously the complexity of such device is unrealistic, and this offers no biological plausibility.

HNNs are fully interconnected neural networks, thus relying on a complete graph, with no loops, and in which connections are weighted over $\mathbb{Z}$. Formally, a HNN is supported by a graph $\mathcal{H} =< n, w : [|1;n|] \times [|1;n|] \to \mathbb{Z} >$ where:

- n is the number of neurons, each neuron being associated with a unique index,
- $w : [|1;n|]^2 \to \mathbb{Z}$ is the edge weight function.

Note that HNNs are bidirectional - or symmetric -, which means that $\forall i, \forall j \in [|1;n|], w_{ij} = w_{ji}$. Moreover, HNN contain no loops: $\forall i \in [|1;n|], w_{ii} = 0$. The number of connections in a HNN with n neurons is therefore $\binom{n}{2} = \frac{n(n-1)}{2}$.

The model for $n = 8$ neurons is depicted in Figure 2.4.

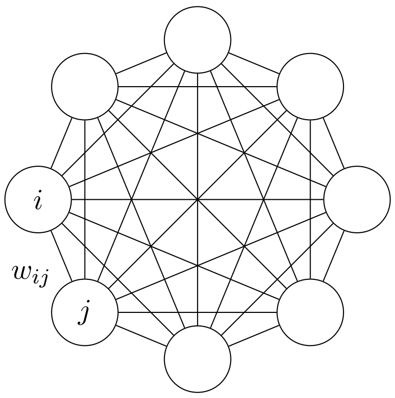


Figure 2.4: 8 node-Hopfield network model. All nodes are connected to each other through a total of 28 bidirectional edges.

2.4.2 Learning and retrieving

There are two behaviours for the HNN dynamics corresponding to the two models described in the previous section. The first one considers that neurons are updated asynchronously - that is independently and sequentially - an iteration being achieved when all neurons have been updated. In the second one, all neurons are updated at once in a synchronous way. In both cases, HNN convergence is assured through iterations. At iteration t, the value of the i-th neuron is given by v_i^t. Given initial conditions $\left(v_i^0\right)_{1\le i\le n}$, and if the update is asynchronous, each iteration is then assured using the following rule:

$$\texttt{for } i \texttt{ from 1 to n:} v_i^{t+1} = \begin{cases} 1 & \text{if} \sum_{j=1}^{n} w_{ij} v_j^t \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

Note that neuron values are binary and in $\{-1;1\}$, with the exception of the initialization where neurons values can be advantageously set to 0. This neutral value would correspond to an erased unknown character. Note also that HNN can be modified to use the alphabet $\{0;1\}$ instead with no impact on performance. Those are not considered in the sequel.

The dynamics correspond globally to a matrix product, which ease some proofs on convergence. In particular, bipartite HNNs always converge [22].

In order to enable the retrieving of randomly generated messages from part of them, associative memories require the number of those messages to be reasonable. If not, an erasure of part of a learned message would likely lead to a retrieving ambiguity. This ambiguity would not be caused by the functioning of the device but by the learning set itself and therefore would be inescapable. Let us illustrate this with the following example: if an associative memory, treating Latin alphabet messages, has learned messages “meat” and “meet” and is asked to retrieve a message from “me*t” where “*” represents an erased character, there is no *a priori* reason to choose one rather than the other.

An HNN with n neurons is able to learn binary messages of length n over $\{-1;1\}$. After the learning of M such messages $\mathbf{d}^1 \dots \mathbf{d}^M$, the weights are given by the following formula:

$$\forall i, \forall j \in [|1;n|], w_{ij} = \begin{cases} \sum_{m=1}^{M} d_i^m d_j^m & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

This learning process implicitly requires each neuron to be associated with a bit position in the messages. More precisely, the i-th neuron corresponds to the i-th bits of the messages to learn. It is interesting to note that the learning of a message or of its inverse would lead to the same effect on the network. Thus one should say a HNN learns both the messages and their inverse. Note also that the connection between two neurons is increased if these have the same value, which is compatible with Hebb's rule.

The retrieval of a learned message $\mathbf{d}^m$ is then assured using the dynamic rule (2.1): a distorted message is presented as an input[6] of the network, with values 0 where bits have been erased. Then the system iterates until reaching a fixed point, that is to say an iteration $t+1$ such that $\forall i \in [|1;n|], v_i^{t+1} = v_i^t$. The obtained persistent values are then mapped reciprocally to a message to read the output decision of the network.

2.4.3 Performance and limitations

A measure of performance of a HNN is to compute the message retrieval error rate as a function of the amount of data erased in the input. Anyway, this does not take into account the number of iterations needed, which is also an important characteristic as far as biological plausibility is concerned.

The number of messages a Hopfield network can learn and retrieve with a good retrieving probability is limited. We call this parameter, usually named capacity in the literature, the diversity in order to fit with the information theory terminology. An upper-bound has already been expressed in [23] as:

$$\boxed{M_{\max}(n) = \frac{n}{2\log(n)}} \tag{2.3}$$

This is a very restrictive upper-bound (by comparison, a network with a single neuron associated with each message to learn through the connections it share with some input neurons would present a better diversity, that is n).

Figure 2.5 depicts the performance measured on HNN for various numbers of learned messages and for $n = 150$. The corresponding bound is $M_{\max}(150) \approx 30$. This performance is measured and displayed using two parameters. The first one is the message error rate (MER), that is to say the measured probability for a learned message with some erased bits not to be retrieved correctly. The second one is the bit error rate (BER), corresponding to the measured probability for a bit in such a message not to be successfully retrieved. Note that the BER is close to 0.5 when the input erasure rate is 1 since it is the probability to guess a uniformly randomly chosen bit. This figure shows that, even without erasure at the input, a number of learned messages less than $M_{\max}(150)$ leads to non-ideal performance.

[6]The input of the network is the initial values of neurons.

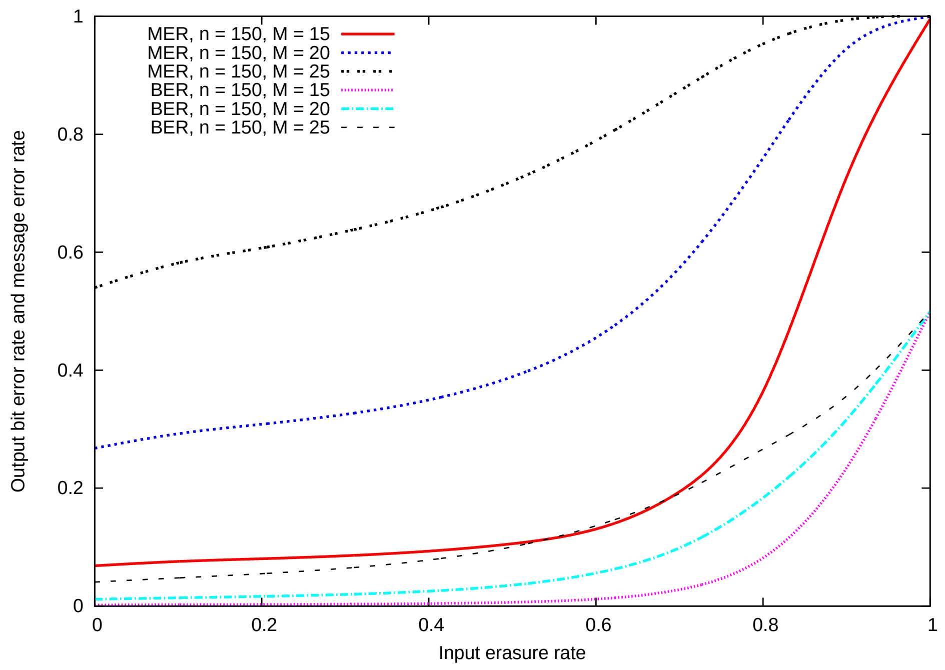


Figure 2.5: Performance in terms of MER (message error rate) and BER (bit error rate) of a HNN with $n = 150$ neurons and 3 different quantities of messages learned (M) as a function of the ratio of erased bits in the inputs.

Moreover, the length of messages must be equal to the size of the network. It is therefore impossible to learn many short messages on such networks.

As Hopfield networks with n neurons learn binary messages of length n, (2.3) states that the total amount of binary data learned, called the capacity, is bounded by:

$$\boxed{C_{\max}(n) = \frac{n^2}{2\log(n)}} \tag{2.4}$$

This equation is not really surprising as HNN are learning messages on connections and those are growing quadratically with the size of the network (n).

Given a HNN after the learning of M messages, one can easily derive from Equation (2.2) that the connection values are in $\{2m - M, 0 \leq m \leq M\}$ and can therefore be encoded over $\lceil \log_2(M+1) \rceil$ bits. A HNN with n neurons contains $\dfrac{n(n-1)}{2}$ such connections. So the memory used by a HNN with n neurons after the learning of $M_{\max}(n)$ messages is:

$$Q^{\max}_{used}(n) \approx \frac{n(n-1)\log_2(M_{\max}(n)+1)}{2} \tag{2.5}$$

The efficiency is then obtained by comparing the amount of learned data with the total amount of memory used with still retrieving capabilities. For the HNN, it is expressed by the following formula:

$$\boxed{E(n) = \frac{C_{\max}(n)}{Q^{\max}_{used}(n)} \approx \frac{1}{\log(n)\log_2(\frac{n}{\log(n)})}} \tag{2.6}$$

Note that this efficiency tends to 0 as the number of neurons tends to infinity. This limitation is due to two factors: the edge weights that are growing with the number of learned messages, and the loss in efficiency of the retrieving process as the network size increases. This limitation is problematic as a HNN must contain a large number of neurons to learn many messages. Note that some previous works have shown it is possible to increase the diversity of HNN; those nevertheless require a larger amount of information used and do not lead to a better efficiency than the HNN [20, 24]. Note also that the HNN weights are likely to be close to 0 after the learning of many messages. Yet it is not possible to prune weights as in some other neural networks targeting classification [25] without the cost of a large decrease in performance.

Note that the notion of efficiency can be tricky. At first sight, it seems to be upper-bounded by 1. Actually it can be larger than 1 in some cases as it does not consider the ordering of messages (learning m messages of 1 bit is different for instance from learning a single message of m bits).

To illustrate this property, let us consider the following toy example: the learning of M randomly generated messages of 1 bit. As we do not consider ordering, there are only four possibilities after such a learning: either no message has been learned, or just message “0” has been learned, or just message “1” has been learned or both messages “1” and “0” have been learned. Thus we can simply encode the result on two bits (for instance one bit to encode whether the message 0 has been learned or not and one bit to encode whether the message 1 has been learned or not). The efficiency of this encoding is:

$$E(m) = \frac{M}{2} \tag{2.7}$$

It is clearly possible to make this particular efficiency arbitrarily large. One could object that this result comes from the fact some messages are learned a large number of times. Actually a learning device could benefit from this non-ordering aspect even if all messages are distinct one from another. Consider the example in Appendix A for more details.

More precisely, there exists a link between the number of learned messages, their size, and the optimal efficiency. As given by the previous example, one can easily show that if the number of messages is large compared to two to the power of their size, the efficiency can be made arbitrary large. On the other hand, considering a single message of size n, obtained with an *i.i.d.* variable, the efficiency is limited to one (universal source compression).

One can figure out a pretty good approximation of the optimal efficiency considering the following construction. A set of ordered messages can be obtained from a set of unordered ones if their relative address is encoded directly inside them. Considering M messages, this requires adding up to $\log_2(M)$ bits to each message. Thus, given n and M, an upper-bound on efficiency is:

$$E_{\max}(n, M) = \frac{n + \log_2(M)}{n}$$

This is an approximation of the optimal efficiency as one could find a better way to retrieve order from messages. However, some results presented in this document are very close to this bound, showing its quality. Figure 2.6 depicts this bound for different values of m and n.

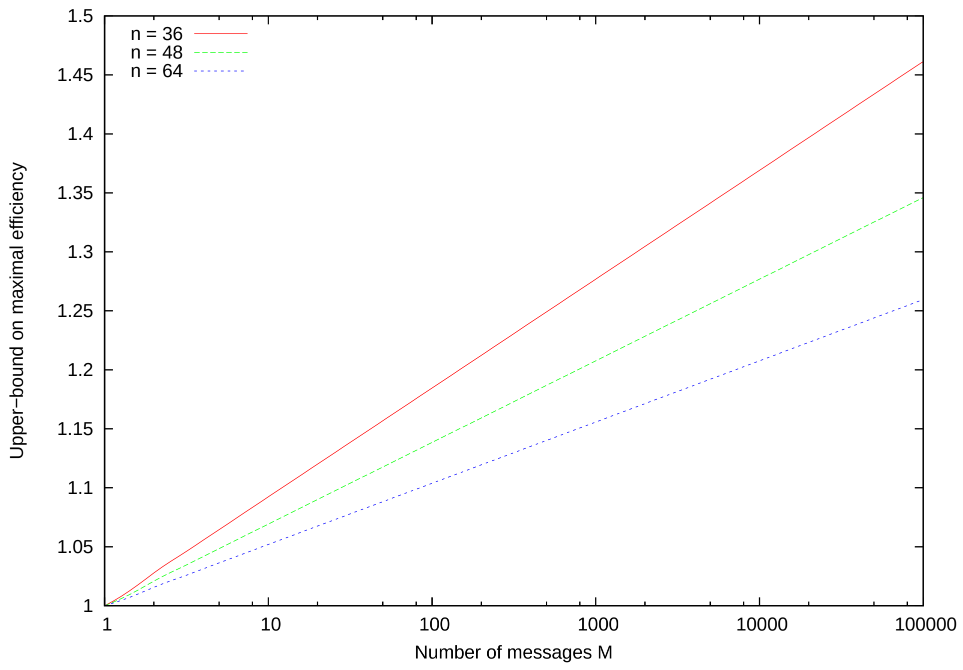


Figure 2.6: Bound on the maximal efficiency for different values of n an m.

Another drawback of the HNN is that with its size n grows the number of iterations needed for each retrieval. In the case of an asynchronous network, this number of iterations is unreasonable (up to several hundreds for a network with several hundreds of neurons). Figures 2.7 and 2.8 depict the evolution of the number of iterations in case of respectively an asynchronous and a synchronous HNN, as a function of the number of learned messages for various learning set sizes. We can see that random input messages also eventually converge but require a larger number of iterations.

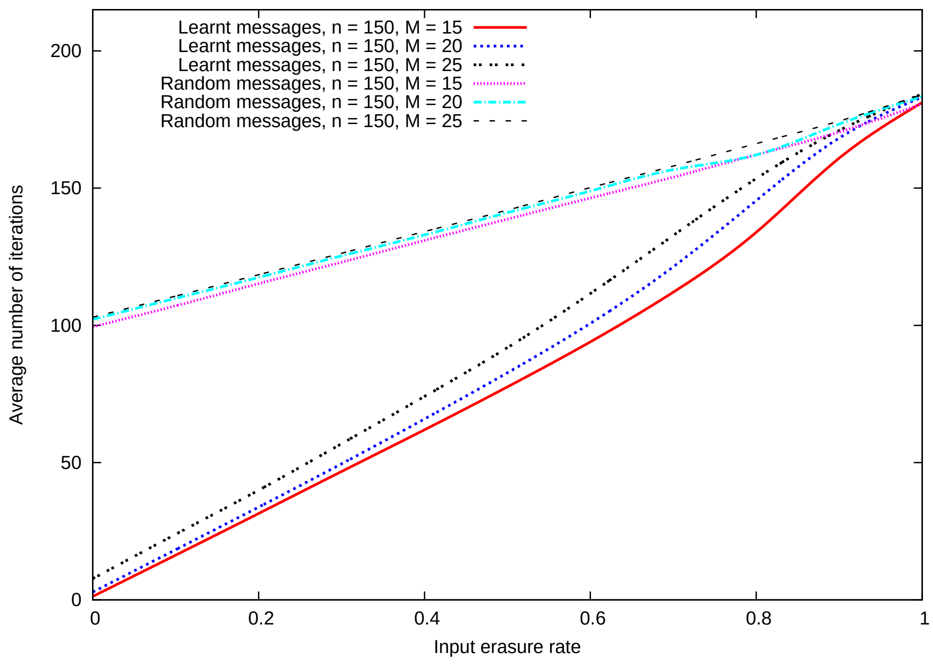


Figure 2.7: Average iterations of a asynchronous HNN with $n = 150$ neurons and 3 different numbers M of learned messages.

In conclusion, HNNs propose a simple way to learn and retrieve messages using a neural implementation. However, their efficiency tends to 0 as their size grows and the number of messages they can learn is sub-linear. The next chapters will present how to combine neural networks and error correcting codes to transcend these limitations.

In this chapter, we have introduced different tools: graphs and associative memories. We saw that the performance of the latter is limited in the case of the HNN: both diversity and capacity are far from being optimal, not to mention the lack of biological plausibility of the architecture.

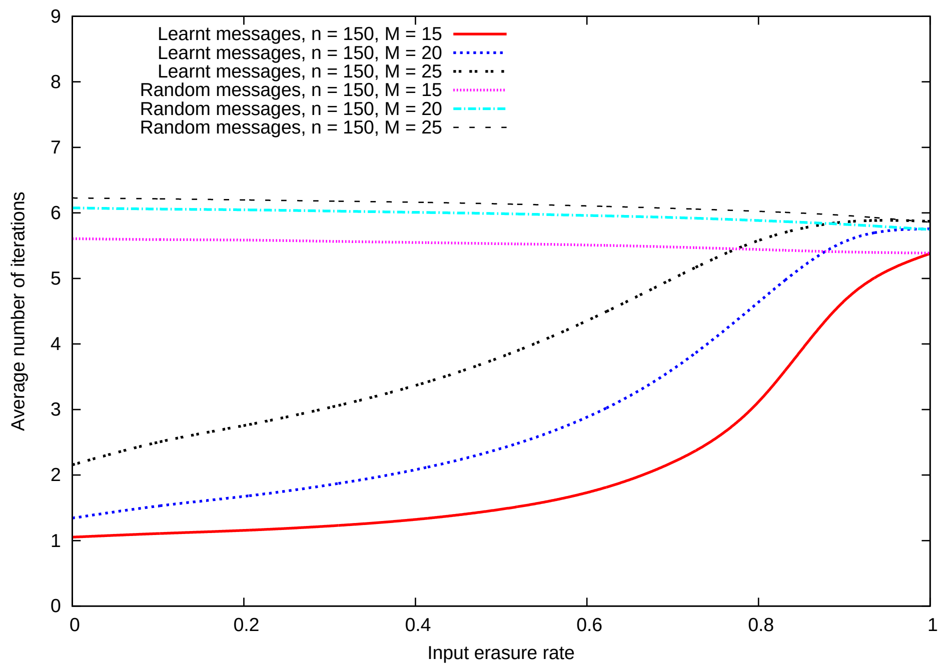


Figure 2.8: Average iterations of a synchronous HNN with $n = 150$ neurons and 3 different numbers M of learned messages.

The notions of diversity, capacity and efficiency are going to be at the very heart of all comparisons in this document. Moreover, we proved that efficiency can surpass the value of 1*, giving a harder challenge for models being introduced in the next chapters.*

3 Recurrent neural networks

Contents

There are two families of neural networks: the feedforward and the recurrent ones. The former have been particularly studied as they are more suitable for a mathematical analysis. On the other hand, distributed codes, which we are going to introduce into neural networks in the next chapter, rely on recurrent graphs.

In this chapter, we present different candidates for the support of information in recurrent neural networks. We then present how to use cliques to build associative memories more efficient than HNNs.

3.1 Recurrent networks

3.1.1 Cycles, populations and cliques

3.1.1.1 Cycles

An essential notion discussed in this document is that of a cycle. Cycles in graphs are paths such that the starting and finishing points are the same. If the graph is bidirectional, these paths should not use the same edge twice (in particular cycles in bidirectional graphs contain at least three edges).

More formally, let us consider a graph $< \mathcal{V}, w >$. We recall that $\mathcal{V}$ is the set of vertices and w the weight function over the edges. A path in this graph is a sequence $[v_1; v_2; \ldots; v_l]$ such that $\forall 1 \leq i < l, w_{v_i v_{i+1}} \neq 0$, that is to say a sequence of vertices such that each one is connected to the next one. A cycle is a path $[v_1; v_2; \ldots; v_l]$ such that $v_l = v_1$. Let us point out that if $[v_1; v_2; \ldots; v_l]$ is a cycle, then $\forall d \in [|1; l-1|], [v_{1+d \pmod l}; v_{2+d \pmod l}; \ldots; v_{l+d \pmod l}]$ is also a cycle. To avoid any ambiguity, this set of cyclic permutations of cycles will be considered to be an unique one. In other terms, cycles have neither starting nor ending vertices.

Figure 3.1 shows an example of a cycle in a directed graph. Note that this would remain a cycle if the graph was bidirectional.

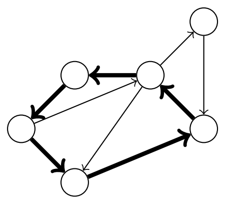


Figure 3.1: Example of a directed graph containing several cycles. One of them is emphasized using bold connections.

An elementary cycle is one that does not contain any sub-cycle with the exception of itself and the empty cycle. This means that it does not contain the same vertex twice. Note that this definition is reminiscent of prime numbers. Moreover, any cycle can be decomposed in a unique way as a list of elementary cycles.

Let us insist that the cycles entities are defined without consideration for weights.[1]

In a fully interconnected graph (complete graph) with n vertices, it seems at first glance that there are as many cycles of length l (containing l vertices) as tuples of vertices of length l, that is $\frac{n!}{(n-l)!}$.

[1]This means that two graphs $< \mathcal{V}, w_1 >$ and $< \mathcal{V}, w_2 >$ share the same cycles if $\forall v_1, v_2 \in \mathcal{V}, w_1(v_1, v_2) = 0 \Leftrightarrow w_2(v_1, v_2) = 0$.

Actually, some cycles are counted several times. By removing these duplicates, the number becomes:

$$C_l^n = \frac{n!}{l(n-l)!} \tag{3.1}$$

This leads to the maximum number $\mathcal{B}_c(n)$ of elementary cycles (of any length) in a graph of n vertices:

$$\mathcal{B}_c(n) = \sum_{l=2}^{n} \frac{n!}{l(n-l)!} \tag{3.2}$$

Obviously, this number is astronomically large. For instance, a graph with $n = 100$ vertices may contain more than 10^{150} distinct cycles.

However, thus far, cycles represent static properties. According to the model considered, a cycle will not necessarily have a predictable temporal behaviour such as for instance an infinite recurring activity on it. Moreover, cycles are not resilient entities: the loss of a single vertex can result in the loss of the entire cycle. Finally, in most models the dynamic activity of a cycle is reduced, at a given time, to that of one of its neurons. Thus intertwined cycles would likely complicate the retrieval of stored pieces of information.

In order to efficiently store pieces of information into our networks, we will consider another grouping entity: populations of neurons.

3.1.1.2 Populations

Let us define a population of neurons as a set of neurons that activate infinitely often given a certain impulse on the network. Let us point out that contrary to the definition of cycles, this is a non-static one.

This terminology of populations is found in the neurobiological literature [26–28], where populations of neurons are sets of neurons that activate at the same time - one could also use the term population coding. It is most of the time associated with the notion of synchronisation.

In realistic models, there exists a link between populations and cycles. More precisely, let us make the assumption that a neuron does not activate infinitely often if its inputs surpasses the threshold just once. Then the following theorem stands:

Theorem 3.1.1 *A cycle-free graph has no population of neurons.*

Indeed, one can prove this by contradiction: if a temporary input produces an infinite activation of a neuron then this neuron receives inputs infinitely often. This means that there exists a neuron connected to it which also activates infinitely often. As there are a finite number of neurons, one will eventually loop back to a previously considered neuron by iterating this principle, building a cycle in the graph. □

However, the notion of population is defined on temporal activities and their characterization can turn difficult. Actually, even minimalist models for neural networks may lead to chaotic behaviors, preventing any simulated way to find them. Yet populations can be fairly accurately inventoried through simulation, contrary to cycles.

There may be as many populations as there are subsets of neurons in the graph, that is to say 2^n, which is still astronomically large. Nevertheless, actual simulations with random neural networks based on physical observations lead to a much more smaller number.

Figure 3.2 shows the evolution of the number of estimated populations in a randomly generated typical neural network as a function of a parameter T. This parameter is the number of times a neuron has to activate to be considered as being part of a population. Thus, the quality of the approximation of populations increases with T. Neurons are randomly generated using parameters that fit biological observations: they have multiple characteristics such as relaxation time about 10^{-3}s, exponentially decreasing outputs, various thresholds, a proportion of 20% of inhibitive neurons (which harden the activation of others). Twelve of them are input neurons which are initially activated or not. The network is based on a small-world construction and coloured by time propagations according to an euclidean distance, weights and directionality. A neuron is considered to be part of a given population if the number of times it switch as activated is more than T. The populations are computed this way for all possible inputs and the figure draws the number of different outputs thus obtained. For instance, for a number of times equal to 100, it shows that among the 4096 possible inputs, about 500 different populations were observed. After a reasonable simulation time estimated populations seem to converge to a few ones. Note that the curve is not decreasing with T since it does not represent the number of neurons that are part of populations but the number of populations with no consideration for their size.

However, the number of measured populations is dramatically smaller than expected. This phenomenon can be eventually explained: each population relies on a kind of a self-persistent engine. This engine is based on a recurrent entity,

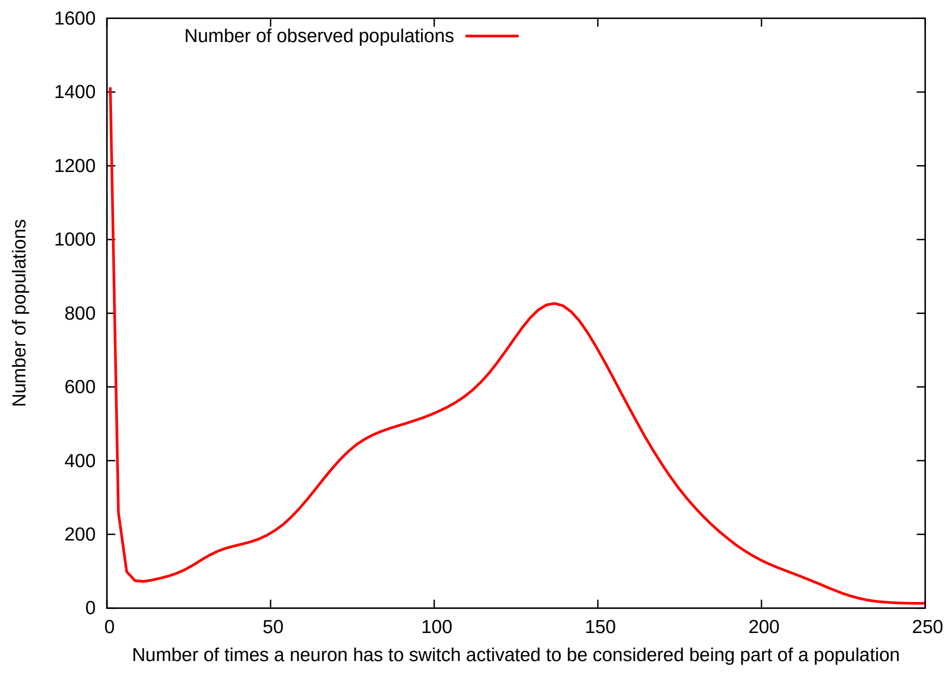


Figure 3.2: Number of observable populations as a function of the number of times (T) a neuron must activate to be considered being part of one in a typical neural network with $n = 144$ neurons.

which is self-providing. As those engines are self-persistent, they are likely to be disjoint. If not, the activation of one would lead to the activation of another every time, making them indistinguishable. In such conditions, there can only be a sub-linear number of populations.

Thus, populations and cycles are too difficult to control. In order to store information in neural networks, we will have to consider more efficient and controllable entities.

3.1.1.3 Cliques

One can refine the notion of cycle by considering cliques. A clique in a graph is a set of vertices such that each one is connected to each other. Due to this definition, it is much simpler to consider a symmetric graph, meaning that if there exists a connection from vertex n_1 to vertex n_2, a connection with the same properties links vertex n_2 to vertex n_1. Figure 3.3 draws an example of a clique in a symmetric graph.

Characterizing cliques is much simpler than characterizing cycles. This as-

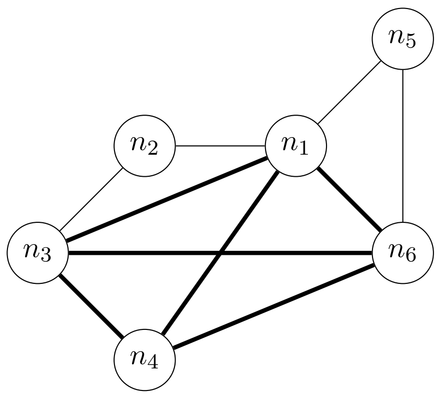


Figure 3.3: Example of a clique in a graph. The clique is characterized by the bold connections.

pect will be described in section 3.3.1.1. Obviously, the maximum number of cliques in a graph with n nodes is 2^n, corresponding to each subset of neurons.

Note that a clique defines $B_c(l)$ cycles since the sub-graph containing only this clique is complete.

Finally, one could insist that a clique is a strong entity, which means that any subset of nodes of a clique defines another clique. A clique that is not a sub-clique of a bigger one will be called a maximal clique.

Thus cliques define different groupings of neurons in a neural network. The number of cliques in a graph can be astronomically large. Moreover, as we will detail this in Chapter 5.1, they give very redundant and resilient properties to the network, making them good candidates to store pieces of information in a neural network.

3.2 Importance of cycles

Cycles, through cliques, are excellent candidates for the storage of information. The question addressed in this section is: reciprocally, does an efficient network requires cycles? We show that a neural network without cycles (and therefore without populations nor cliques) would not be reasonable as far as computing aspects are concerned.

Considering a computational neural network, there are two reasons why cycles have to be introduced. The first one is related to computing aspects: there must be regions in the networks associated with the semantic of the `while` operator of common programming languages. The second one is associated with the variables manipulated by the program. Indeed, if a variable is manipulated by the program then it is likely to be accessed and modified at several steps of the program. Two cases are then possible: either the location of a variable is at most used once, meaning that the program needs linear space as a function of the number of times a variable is accessed, or there is a cycle in the network.

But is there a way to produce an effective computational neural network without cycles?

Let us consider a program trying to find a language known to be of `NP-complete` complexity. And let us make the conventional assumption that $\mathtt{NP} \neq \mathtt{P}$. If the neural network obtained to simulate this program does not contain any cycles, then each neuron will be visited at most once. So, if we use another Turing machine to simulate this obtained network, the complexity, in terms of number of elementary operations, will not exceed the number of neurons (probably multiplied by a constant factor). As the problem is known to be in `NP`, the number of neurons is at least growing exponentially with the entry of the program.

This means that a neural network as considered in this section requires an exponential number of neurons to solve `NP-complete` problems if no cycles are allowed.

Thus, it appears reasonable to make the hypothesis that high order neural networks (able to store and use pieces of information) should contain cycles. Added to the fact that cycles are a very interesting candidate for storing pieces of information, this result gives the tone of the rest of this document: how is-it possible to use cycles to store more pieces of information in neural networks?

3.3 Designing networks from virtual cliques

In this section, we introduce an original way to design neural networks from virtual cliques. Later, in Chapter 5, we will see how to use real cliques as support of information in neural networks.

3.3.1 Bipartite association

3.3.1.1 Model

We want to associate pieces of information with cliques in neural networks. To achieve this, let us project neural networks onto another graph where cliques are modelled as nodes.

Consider a symmetric bipartite graph $\mathcal{G} =< (\mathcal{V}_1 \cup \mathcal{V}_2), w >$ where:

- $\mathcal{V}_1$ and $\mathcal{V}_2$ are two disjoint sets of vertices,
- $w : \mathcal{V}_1 \times \mathcal{V}_2 \to \mathbb{Z}$ is an edge weight function.

The graph is coloured using a threshold function $T : \mathcal{V}_1 \cup \mathcal{V}_2 \to \mathbb{Z}$.

The dynamic function f_n for node n, as defined in Figure 2.3, is set to:

$$f_n : v \to \begin{cases} 1 & \text{if } v \geq T(n) \\ 0 & \text{otherwise} \end{cases}$$

This means that a node will dynamically be activated if the sum of its input surpasses its threshold.

Applied to our abstract representation of cliques in neural networks, one can map set $\mathcal{V}_1$ to neurons and set $\mathcal{V}_2$ to the cliques in the same network. With this association, an impulse on the network will activate several cliques. After a few iterations, it will eventually reach a fixed point. If one can associate messages with cliques, then this fixed point would be the learned message corresponding to the impulse. Let us call such a network a clique-node network.

Let us denote $\mathbf{n}^t$ and $\mathbf{c}^t$ the binary vectors representing respectively the state of the neurons in $\mathcal{V}_1$ and $\mathcal{V}_2$ at step t. $\mathbf{n}^0$ is the impulse of the network, that is to say the neurons originally activated. Formally, given an impulse the dynamic of the network is defined as:

1. $\forall j \in \mathcal{V}_2, c_j^{t+1} \leftarrow f_j(\sum_{i \in \mathcal{V}_1} n_i^{t+1} w(i,j))$
2. $\forall i \in \mathcal{V}_1, n_i^{t+1} \leftarrow f_i(\sum_{j \in \mathcal{V}_2} c_j^{t} w(i,j))$

Figure 3.4 represents a bipartite graph associated with the graph depicted in Figure 3.3. Three maximal cliques have been identified. One may note that, more generally, with any graph can be associated a unique clique-node network, where only its maximal cliques are represented.

3.3.1.2 Performance

Theorem 3.3.1 *Given a bipartite network as previously described and an impulse, the network will eventually converge to a fixed point.*

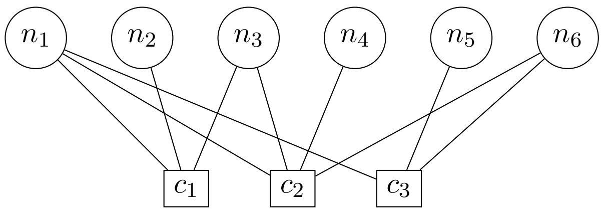


Figure 3.4: Bipartite graph representing the association between maximal cliques and nodes of the graph depicted in Figure 3.3. Cliques are squares and nodes are circles.

Proof A first result is that the network will eventually reach a state from which it will always loop back to it (such a state is called an attractor in non linear dynamic systems theory). Indeed, there are only a finite number of different states in the network and therefore after an infinite number of iterations, some will be encountered infinitely often. Moreover, as the state of the system only depends on the previous one (memoryless system), the sequence of states between two consecutive occurrences of such state will also be the same.

Finally, the network diminishes the value of $\mathbf{c}^t W^T \mathbf{v}^t$ where $W_{ij} = w(i,j)$ at each step. As this value is specific to a given state of the network, it necessarily eventually reach a fixed point. □

Figure 3.5 depicts an example of a network which stores eight different messages. These messages have moreover good separability as they can be retrieved in case of two erasures in the impulse.

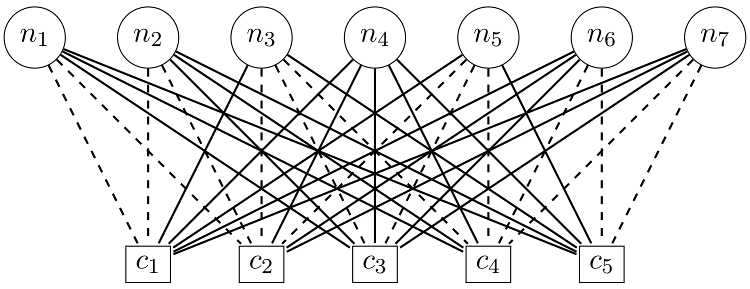


Figure 3.5: Example of a clique-node network with 7 neurons and 5 cliques. This network recognizes 8 different messages and retrieves the closest one in case of two erasures. Note that, as the network contains 5 cliques, it could have recognized up to a maximum of $2^5 = 32$ different messages. Full lines represents weight 1 and dashed ones weight -1. All thresholds for neurons are set to 1 whereas they are set to 2 for cliques.

3.3.1.3 Learning

Let us suppose that one wants to design a clique-node network able to learn m binary messages $\mathbf{d}_1, \ldots, \mathbf{d}_m$ of length k:

Let $< \mathcal{V}_1, \mathcal{V}_2, w >$ be a clique-node network such that $\mathcal{V}_1$ contains k elements, $\mathcal{V}_2$ contains m elements and w is defined by:

$$w(v_i, c_j) = \begin{cases} 1 & \text{if } d_{j_i} = 1 \\ 0 & \text{otherwise} \end{cases}$$

That is to say that one clique is associated per message and is connected to the neurons accordingly to the bits of the message. Thresholds can then be adjusted depending on the targeted application. In particular, if thresholds are set to m and k respectively for neurons and cliques the network realizes a go no-go sort.

So, a network with k neurons and m cliques can learn and retrieve up to m messages of length k.

Figure 3.6 shows the comparison between the HNN and the bipartite model for the same amount of data learned. Note that the bipartite model contains only binary connections and these are much less than in HNN.

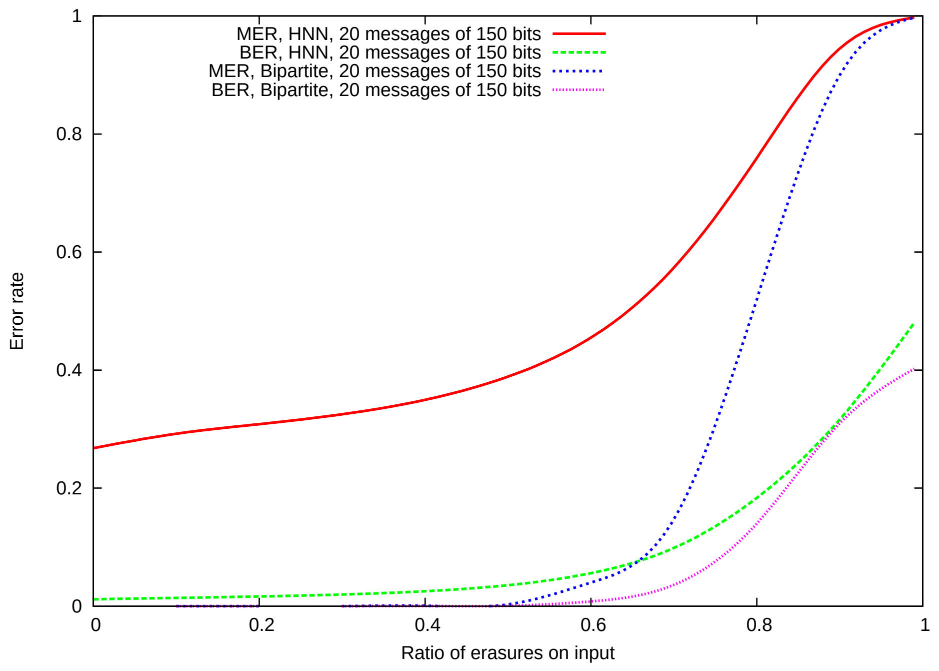


Figure 3.6: Comparison between the HNN and the bipartite model performance for the same amount of data learned. The bipartite model contains many fewer connections (approximatively 7.5 times fewer) and those are binary. Both the MER (message retrieval error rate) and the BER (bit error rate) are depicted.

This model is thus able to learn a linear number of messages, which is slightly better than the Hopfield one. Moreover, it dissociates the length of messages from the size of the network, making it useful for many more applications.

However, that is still disappointing compared to the number of possible cliques in a graph. To exploit a higher potential of cliques in the neural networks, one must find a way to control their interactions.

We have shown in this chapter that cliques are very interesting candidates to store pieces of information in neural networks. Moreover, cycles are necessary components of them as far as computability is concerned. The question remains on how to design such devices. From what we have seen so far, we should try to use entwined cliques, with some limitation since we want to keep control over them.

4

Error correcting codes and neural networks

Contents

There is one domain that has made its speciality of the retrieval of partially erased or erroneous messages: error correcting coding. This field has known an important breakthrough in 1993 with the introduction of turbo codes by Claude Berrou and Alain Glavieux [29] and later with the rediscovery [30] of LDPC codes [31]. Since then, distributive coding has been the center of interest for many scientists of the community. In this chapter, we introduce the notion of error correcting codes. We present some original contributions: the clique code, and an optimal construction for constant weight code that will be used in Chapter 5.

We then present a novel way to decode any error correcting code using a large number of neurons. Finally we present an original way to use these codes to increase the performance of associative memories tenfold using both orthogonal codes and HNNs.

4.1 Introduction to error correcting codes

A classic approach to error correcting codes is to consider them as a function that adds redundancy to initial messages in order to protect them during their transport to a destination. The image of such a message is called a codeword.

An example of a trivial code is the repetition code, which is defined through the function:

$$C : \begin{array}{ccc} \mathcal{A}^* & \to & \mathcal{A}^* \\ m & \mapsto & mm \end{array}$$

Elements in the alphabet $\mathcal{A}$ are called symbols, and $\mathcal{A}^*$ is the set of messages made of symbols in $\mathcal{A}$.[1] Note that, in almost all examples presented in this document, the length k of initial messages is unique as well as the length n of codewords. In other words, codes will be specified by a function:

$$C : \mathcal{A}^k \to \mathcal{A}^n$$

The obtained codewords are then transmitted via a channel, which is characterized by a random function $\tilde{\ }: \mathcal{A}^* \to \mathcal{B}^*$. Note that the channel function may change the alphabet: a typical example is when a binary codeword is transmitted into a Gaussian channel (In such case, $\mathcal{A} = \{0; 1\}$ for instance and $\mathcal{B} = \mathbb{R}$).

[1]We borrow this notation from language theory.

The channel function usually alters the codewords it receives by adding errors or erasures. Later in the section, $\mathcal{B}$ will typically be either $\mathcal{A}$ in case of errors (some symbols are transformed to others) or $\mathcal{A} \cup \{\perp\}$ where $\perp \notin \mathcal{A}$ denotes an erased character.

A code will be said to be adapted to $\widetilde{\ }$ if it is such that one is likely able to retrieve the initial message m given the altered codeword $\widetilde{C(m)}$. In that sense, the previously defined repetition code is not strictly speaking an error correcting code since an ambiguity will result if the channel function changes any single symbol of a codeword (that is: adds an error). Nevertheless, if one considers the function that adds twice the message as redundancy, such errors could be corrected: the double repetition code is an error correcting code.

Figure 4.1 depicts this principle. An initial message (the square on the top) is transformed adding redundancy to it (dashed). Both are transmitted through a transmission channel such that the received signal has been deformed. The decoding process will then try to identify the most likely transmitted codeword, and then hopefully retrieve the initial message.

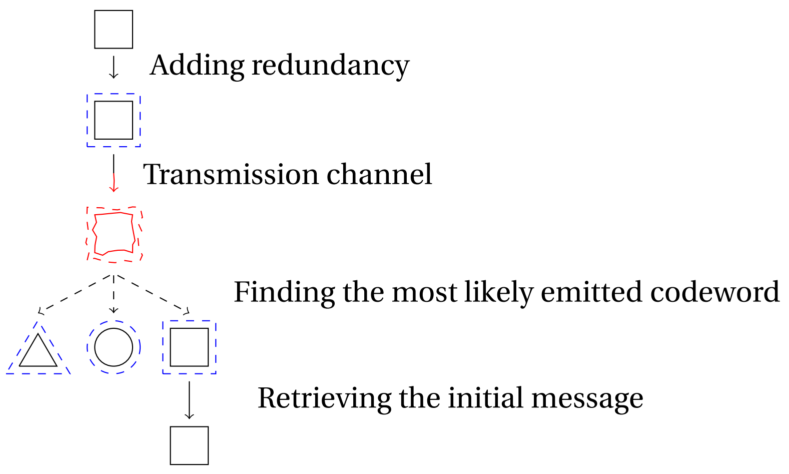

Adding redundancy
Transmission channel
Finding the most likely emitted codeword
Retrieving the initial message

Figure 4.1: Representation of the coding/decoding principle. Redundancy is added (dashed) to an initial message such that even if both are altered through a transmission channel, one will likely be able to retrieve the message.

A more general approach to codes is as follows. The codewords of a code $\mathcal{C}$ are any subset of $\mathcal{A}^*$. Thus, contrary to the previous definition, we do not need to have an explicit function to associate codewords with input messages.

In all cases, one must introduce a topology to perform the decoding, that is to say introduce a metric that can measure the proximity between messages. Considering codewords are all of the same length n, let us define such a metric on $\mathcal{A}^n$

as:

$$\begin{aligned}\forall v_1, v_2 \in \mathcal{A}^n, d_h(v_1, v_2) &= \sum_{i=0}^{n-1} v_{1i} <> v_{2i} \\ \text{where } v_{1i} <> v_{2i} &= \begin{cases} 1 & \text{if } v_{1i} \neq v_{2i} \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

This metric, which counts the number of distinct symbols between two messages, is called the Hamming distance. The Hamming distance represents some kind of separability between messages: a high distance between two messages means that they are very different whereas a distance zero means that they are identical. Note that as a metric, d_h also verifies the triangle inequality and the symmetry properties.

In many cases, the channel function is such that the most likely transmitted codeword is the closest one, according to the metric d_h, to the received message.

In other cases, it is usual to adapt the definition of the distance to hold this property.

4.1.1 Linear codes

Linear codes are a particular family of codes with handy properties.

Let us consider $(\mathcal{A}, +, \cdot)$ to be a $\mathbb{K}$-linear space, $\mathbb{K}$ being a field. Then let us denote by 0 the neutral element of $+$ and endow $\mathcal{A}^n$ with $+$ and $\cdot$. A linear code $\mathcal{C}$ over $(\mathcal{A}^n, +, \cdot)$ is a linear subspace of $\mathcal{A}^n$, that is to say: $\forall c_1, c_2 \in \mathcal{C}, \forall \lambda, \mu \in \mathbb{K}, \lambda \cdot c_1 + \mu \cdot c_2 \in \mathcal{C}$. In particular, 0^n (denoting the vector that contains n zeros) is in $\mathcal{C}$. The dimension of the subspace is denoted k. This means that the code is entirely determined given k linearly independent messages in it.

One can define the weight $w(m)$ of a message m as the number of its coordinates which are non zero:

$$\forall m \in \mathcal{A}^n, w(m) = \sum_{i=0}^{n-1} m_i <> 0$$

As for any messages m_1, m_2 in $\mathcal{A}^n$, it holds that $m_1 = m_2 \Leftrightarrow (m_1 - m_2) = 0$, one can relate weights to the Hamming distance:

$$\forall m_1, m_2 \in \mathcal{A}^n, d_h(m_1, m_2) = w(m_1 - m_2)$$

4.1.2 Minimum Hamming distance

A characteristic of prime interest of a code, whether or not linear, is its minimum distance, that is to say the minimum Hamming distance between two distinct codewords. This distance is directly related to the maximum number of errors the channel function may add such that there is no ambiguity in retrieving the initial codeword.

Formally, the minimum distance of a code $\mathcal{C}$ is:

$$d_{min}(\mathcal{C}) = \min_{c_i, c_j \in \mathcal{C}, c_i \neq c_j} d_h(c_i, c_j)$$

Consequently to the previous remarks, and if $\mathcal{C}$ is a linear code, this minimum distance can also be written as:

$$d_{min}(\mathcal{C}) = \min_{c \in \mathcal{C}, c \notin 0^*} w(c)$$

The maximum number of errors $e_{\max}$ that can be corrected on any codeword, given an infinite computing capacity, is obtained from the minimum distance:

$$e_{\max} = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor$$

Also, such a code can correct $d_{min} - 1$ erasures.

In realistic cases, the channel function is such that the most likely transmitted codeword c_t is the closest to the received one $\widetilde{m}$ according to the Hamming distance:

$$c_t \in \underset{c \in \mathcal{C}}{\operatorname{argmin}}(d_h(c, \widetilde{m})) \tag{4.1}$$

A function that associates a message in $\mathcal{C}$ with the received message according to Equation (4.1) is called a Maximum Likelihood (ML) decoder. Thus, an immediate way to build a ML decoder of code $\mathcal{C}$ is to compute the distance between every element of $\mathcal{C}$ and a received message and then to pick the one (or one of those) that achieves the minimum distance. If several messages are as close as others to the received one, a soft ML decoder would make a more subtle decision than to chose one among them. For instance, it can erase the symbols where there exist a contradiction among the different candidates.

As it is sometimes too complex to implement a ML decoder, one can consider using an approximation. A decoder is basically a function that associates a codeword in $\mathcal{C}$ with any received message $\widetilde{m}$. Of course, the performance of a decoder is to be compared to that of the associated ML decoder. The more similar the two functions are, the more efficient the approximate decoder is.

It raises too many questions to propose a channel function model for the brain as far as biological plausibility is concerned. In order to be comparable with the HNN, we will only consider very simple ones. These functions modify symbols of codewords independently (those channels are called memoryless as they do not depend on what has been previously done). In this section, the modification on symbols will be considered either to be coherent with a symmetric channel as depicted in Figure 4.2 or with an erasure channel as depicted in Figure 4.3. Note that both are such that decoding according to the minimum distance criterion gives the most likely transmitted message.

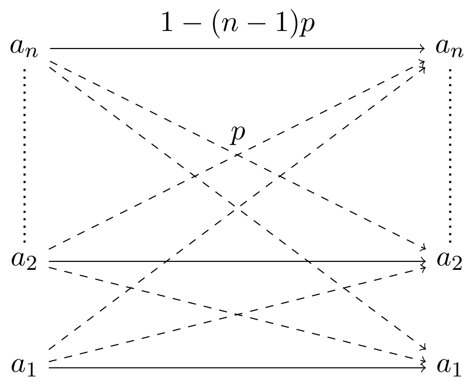


Figure 4.2: Model of the symmetric channel. Dashed transitions represent a unique probability p and straight lines a probability $1-(n-1)p$. Moreover $1-(n-1)p \geq 0.5$ which means it is more likely that a symbol will not be altered.

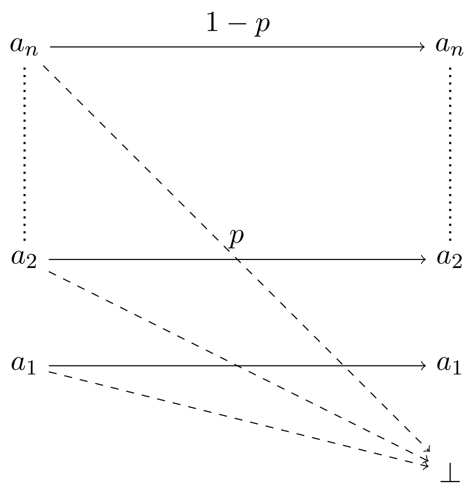


Figure 4.3: Model of the erasure channel. Dashed transitions represent a unique probability p and straight lines a probability $1-p$. Moreover $1-p \geq 0.5$ so it is more likely that a symbol will not be erased.

4.1.3 Merit factor

To estimate the efficiency of a code, one has to compare the amount of redundancy with the correction capability it offers. Without considering this trade-off, it would be immediate and trivial to propose codes with arbitrary large minimum distance.

For instance the repetition code previously defined is not very efficient as its minimum Hamming distance is only 2, which means that it cannot correct any error, while it doubles the size of messages.

A typical example of good code is the extended Hamming code ($n = 8$, $k = 4$ and $d_{min} = 4$). This binary code is defined as follows:

$$C(m) : \begin{array}{rcl} \{0;1\}^4 & \to & \{0;1\}^8 \\ m_1m_2m_3m_4 & \mapsto & m_1m_2m_3m_4r_1r_2r_3r_4 \end{array}$$

where m_i is the i-th bit in m and:

$$r_j = m_j + \sum_{i=1}^{4} m_i \pmod 2$$

It is trivial to show that the minimum distance of this code is 4, which is also the number of bits added as redundancy.

More generally, a good trade-off is to offer a large correcting ability despite a relatively low redundancy. In order to estimate the redundancy added, we introduce the rate r_C of a code $C \subset \mathcal{A}^n$ as:

$$r_C = \frac{\log_{\#\mathcal{A}}(\#C)}{n}$$

A rate value close to 0 corresponds to a lot of added redundancy whereas a value of 1 is obtained when no redundancy has been added.

The merit factor F of a code C is defined as the product of its rate and of its minimum Hamming distance. It has been shown that it has to be greater than one to be interesting in telecommunication applications.

The extended Hamming code previously introduced is such that $F = 2$, the repetition code only achieves $F = 1$.

4.2 Examples

4.2.1 Hadamard codes

For this subsection, let us set $\mathcal{A}$ to be the binary alphabet over $\{-1;1\}$. One can define a recursive family of codes $\mathcal{C}_H(k)$ on $\mathcal{A}^n$ with $n = 2^k$ using the following definition:

$$\mathcal{C}_H(1) = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$
$$\mathcal{C}_H(k+1) = \mathcal{C}_H(1) \otimes \mathcal{C}_H(k)$$

where $\otimes$ denotes the Kronecker product:

$$\forall A \in M_{a,b}, \forall B \in M_{a',b'}, A \otimes B = \begin{pmatrix} a_{1,1}B & \dots & a_{1,b}B \\ \vdots & \ddots & \vdots \\ a_{a,1}B & \dots & a_{a,b}B \end{pmatrix}$$

Thus, the recursive formula of $\mathcal{C}_H$ can be rewriten as:

$$\mathcal{C}_H(2n) = \begin{pmatrix} \mathcal{C}_H(n) & \mathcal{C}_H(n) \\ \mathcal{C}_H(n) & -\mathcal{C}_H(n) \end{pmatrix}$$

The code of length n is then made of the lines of $C_H(k)$ and will be mingled with this matrix. This family forms the Hadamard codes. They have interesting properties: for instance, one can prove by induction that $\mathcal{C}_H(k)$ has minimum Hamming distance $\frac{n}{2}$.

Moreover, they present orthogonal properties. Let us endow $\mathcal{A}^n$ with the sum and product over $\mathbb{R}$. One can define the euclidean scalar product as:

$$\forall m_1, m_2 \in \mathcal{A}^n, < m_1, m_2 >= \sum_{i=1}^{n} m_{1i} m_{2i}$$

Given this scalar product, one can easily shows once again by induction that the family $\mathcal{C}_H(n)$ is orthogonal, which means that:

$$\boxed{\forall c_1, c_2 \in \mathcal{C}_H(k), c_1 \neq c_2 \Rightarrow < c_1, c_2 >= 0}$$

Moreover, the rate of the code $\mathcal{C}_H(k)$ is:

$$r_{\mathcal{C}_H(k)} = \frac{k}{n}$$

which leads to a merit factor:

$$F = \frac{k}{2} = \frac{\log_2(n)}{2}$$

4.2.2 Constant weight codes

Another singular family of codes are the constant weight codes [32]. For better readability, we will only introduce binary constant weight codes. A binary constant weight code of length n, weight w and overlapping r is a code $C_{cw}(n, w, r)$ such that every codeword is a binary (over $\{0; 1\}$) message of length n, containing exactly w "1"s, and such that two distinct codewords cannot share more than r "1"s at the same locations.

Note that due to the structure of the code, the Hamming distance between two codewords is necessarily even. Moreover, the constraints of the codes force the minimum Hamming distance to be larger than $d = \max(2(w - r), 0)$.

4.2.2.1 Weight one

A very particular subfamily of such codes is the one with weight $w = 1$. As a direct consequence, the parameter r is useless.

To obtain the best merit factor, constant weight codes with $w = 1$ must consider all the possible messages (as it increases the rate of the code). For instance, with length $n = 3$ and weight $w = 1$, the optimal binary constant weight code, as far as the merit factor is concerned, is $\{100, 010, 001\}$.

Those codes are obviously not efficient, offering a factor merit $F = \frac{2\log_2(n)}{n}$. On the other hand, they are in most cases ML decodable with a low computing complexity.

Consider for instance that the channel function is such adds a positive random value to each bit of the codeword. Formally, the channel function is such that $\tilde{}: c \mapsto m$ where $\forall i, m_i = c_i + X$, X being an i.i.d. random variable. The most likely transmitted codeword is then the one with its "1"s where the received message has its largest value. Figure 4.4 depicts this process. Among all the values, only the one reaching the maximum is kept.

This decoding corresponds exactly to the winner-take-all rule previously introduced.

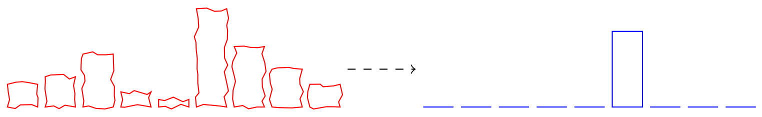


Figure 4.4: A representation of the principle of the constant weight code decoding with $w = 1$.

4.2.2.2 Overlapping one

A more general case is when $r = 1$ which means that two words cannot share more than a single "1" at the same location. It is possible to find an upper-bound on the number of codewords such a code can contain. This upper-bound is given by:

$$\boxed{U^b_{C^{(}_{cw}n,w,1)} = \frac{n(n-1)}{w(w-1)}} \tag{4.1}$$

> *Moreover, if w is a prime number, and if $\exists p, n = k^p$, this upper-bound can be achieved by some codes. See Appendix B for more details.*

4.2.3 Cliques

A last example of error correcting codes is the c-clique code. We introduce this code as it will be at the very foundation of the sparse network introduced in Chapter 5. This is one of the main contributions of our work.

Let us consider a complete graph with n vertices and no loops. This graph defines a code which consists of every c-clique in it.

We first point out that this code can be seen as a particular constant-weight code. Indeed, let us suppose that connections are indexed from 1 to $N_c = \frac{n(n-1)}{2}$. Any c-clique is made of $S_c = \frac{c(c-1)}{2}$ distinct connections, and therefore can be encoded as a codeword of length N_c and of weight S_c containing "1" at the locations corresponding to its connections. However, this choice of representation would lead to a very low code rate and therefore a very low merit factor. A more efficient representation, with the same minimum Hamming distance, is developed thereafter.

In this graph, a clique can be uniquely identified given $\lceil \frac{c}{2} \rceil$[2] distinct well chosen connections (in such a way that every vertex of the clique is specified by one

[2] $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

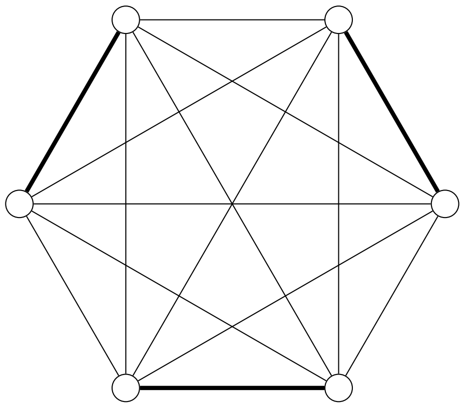


Figure 4.5: Example of a clique with 6 vertices that can be uniquely identified given only 3 connections (in bold). The clique is composed of a total of 15 connections.

connection: see Figure 4.5). If the number of vertices n is large enough, there is not much of a difference between considering distinct connections or not. Moreover, if c is small, there is also not much of a difference between considering ordering connections or not and thus the number of c-cliques in a graph can be approximated by $N_c^{\lceil \frac{c}{2} \rceil}$.[3]

To represent a clique, we can simply consider a codeword of length S_c which contain all the connections of the clique. Still with the hypothesis that n is large enough and c is small, this leads to the fact that c-cliques can be encoded over $\log_2\left(N_c^{S_c}\right)$ bits (which is in good cases much less than the N_c bits of the constant weight representation).

Thus, the rate of the c-clique-code is approximated as:

$$r_{cliques} \approx \frac{\log_2(N_c^{\lceil \frac{c}{2} \rceil})}{\log_2\left(N_c^{S_c}\right)} \approx \frac{1}{c-1} \tag{4.2}$$

Moreover, the minimum distance (in terms of the number of connections) between two cliques is reached when they have almost all the same vertices, with the exception of one which has moved. This corresponds to the following number of connections:

$$d_{\min,cliques} = 2(c-1) \tag{4.3}$$

[3]It is possible to obtain the same result considering the following reasoning: there are $\binom{n}{c}$ c-cliques in the graph. When c is small and n large, this number is close to n^c. As there are about $N_c = n^2$ connections in the graph, this number can also be written $N_c^{\frac{c}{2}}$.

(see Figure 4.6). Thus, the merit factor is:

$$M_{cliques} = r_{cliques} d_{\min,cliques} \approx 2 \tag{4.4}$$

when n is large enough.

This means that c-cliques, for c small in a large graph, form an efficient error correcting code.

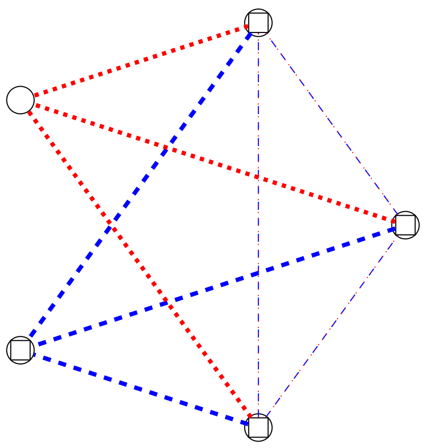


Figure 4.6: Minimum Hamming distance (in terms of connections) between two 4-cliques (cliques of 4 vertices) in a graph. The first one is represented by squares and dashed connections, the second one by circles and dotted connections. The difference (thick) is $2(4-1) = 6$ connections.

4.3 Decoding linear codes using neural networks

4.3.1 A first approach

A neural decoder is a neural network associated with a code $\mathcal{C}$ such that given an impulse corresponding to a received message $\widetilde{m}$, it produces an output corresponding to a decoding of $\widetilde{m}$ in $\mathcal{C}$. One can easily build neural decoders which achieve ML performance in some cases.

If the alphabet is well chosen, the Hamming distance computation between a received message and a codeword can be expressed as a scalar product.

Let us consider a code over $\{-1;1\}^n$. Depending on the channel function, some symbols in messages can be erased (that is: transformed to $\perp \notin \{-1;1\}$). Let us consider that in such case, $\perp = 0$. Thus, either $\mathcal{B} = \{-1;1\}$ or $\mathcal{B} =$

000000
001110
010101
011011
100011
101101
110110
111000

Table 4.1: Code made of 8 codewords of length 6. The minimum Hamming distance is 3. 0 represents the value -1 for a better readability.

$\{-1; 0; 1\}$. Then the following equations hold:

$$\begin{aligned} \forall m \in \mathcal{B}^n, D_{\mathcal{C}}(\widetilde{m}) &= \underset{c\in\mathcal{C}}{\operatorname{argmin}}(d_h(c, \widetilde{m})) \\ &= \underset{c\in\mathcal{C}}{\operatorname{argmax}} < c, \widetilde{m} > \end{aligned} \tag{4.5}$$

Based on this remark, one can set up a neural decoder with good performance. Such a network is built this way: let us consider a bidirectional bipartite graph $< \mathcal{N}, \mathcal{C}_{\mathcal{N}}, w >$ where:

- $\mathcal{N}$ is a set of n neurons and is provided with an indexed one to one function: $I_n : [|1; n|] \to \mathcal{N}$,
- $\mathcal{C}_{\mathcal{N}}$ is a set of $\#\mathcal{C}$ different neurons also provided with a one to one function: $I_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}_{\mathcal{N}}$,
- $w : \mathcal{N} \times \mathcal{C}_{\mathcal{N}} \to \{-1; 1\}$ is such that $\forall n_i \in \mathcal{N}, \forall c_n \in \mathcal{C}_{\mathcal{N}}, w(n_i, c_n) = I_{\mathcal{C}}^{-1}(c_n)_{I_n^{-1}(n_i)}$.

This means that connections in the bipartite graph correspond to the codewords of $\mathcal{C}$. More precisely, the inputs of a neuron in $\mathcal{C}_{\mathcal{N}}$ are exactly the bits of the associated codeword in $\mathcal{C}$. This way, the input of the functions f defined for each neuron in $\mathcal{C}_{\mathcal{N}}$ is the scalar product of the values of neurons in $\mathcal{N}$ and the codewords of $\mathcal{C}$. Figure 4.7 gives an example of the neural decoder associated with the code specified in Table 4.1.

Given Equation (4.5), the ML decoding of a received message $\widetilde{m}$ is a codeword maximizing the scalar product.

Therefore, to decode a received message $\widetilde{m}$ using the proposed neural decoder, one just needs to project $\widetilde{m}$ onto the neurons in $\mathcal{N}$ and then select a neuron in $\mathcal{C}_{\mathcal{N}}$ which achieves the maximal input value. Then, in order to read the corresponding selected codeword, a successive iteration from the codeword neurons

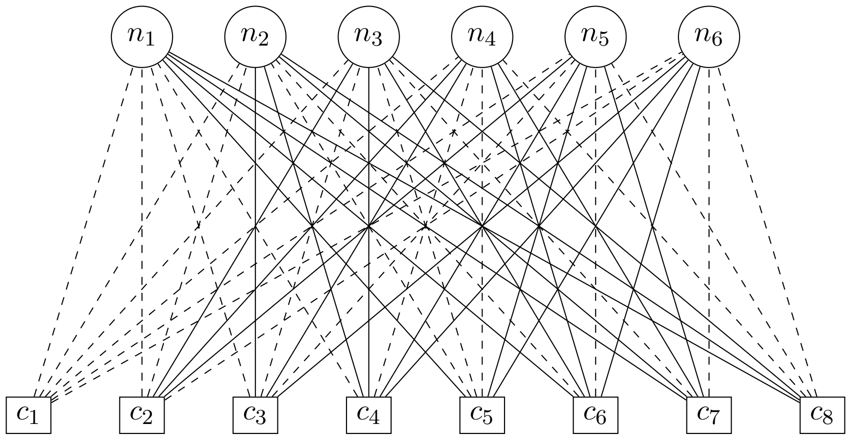


Figure 4.7: Neural decoder architecture associated with the code defined in Table 4.1. Input values are represented by circles and codewords by squares. Solid lines represent weight 1 and dashed lines represent weight -1.

back to the neurons in $\mathcal{N}$ will print the codeword to the input where was originally the received message $\widetilde{m}$.

Selecting the correct neuron in $\mathcal{C}_{\mathcal{N}}$

The decoding process thus works thanks to a selection of the neuron with the maximum input among $\mathcal{C}_{\mathcal{N}}$. Actually, it is not necessary to provide the network with a winner-take-all implementation: a simple trick on the neural function enables this unique selection. In order to do this, let us define the functions $f_{\mathcal{C}}$ for all neurons in $\mathcal{C}_{\mathcal{N}}$ as:

$$f_{\mathcal{C}}(v) = \begin{cases} 1 & \text{if } v > n - d_{\min}(\mathcal{C}) \\ 0 & \text{otherwise} \end{cases}$$

Moreover, the functions associated with the other neurons in $\mathcal{N}$ are set to:

$$f_{\mathcal{N}}(v) = \begin{cases} 1 & \text{if } v \geq 1 \\ 0 & \text{if } v = 0 \\ -1 & \text{otherwise} \end{cases}$$

Given a received message $\widetilde{m}$, there can be only one neuron in $\mathcal{C}_{\mathcal{N}}$ that activates. To prove this latest property, let us first point out that if a neuron $c_n \in \mathcal{C}_{\mathcal{N}}$

is activated, then:

$$\begin{aligned}(<\widetilde{m}, I_{\mathcal{C}}^{-1}(c_n)>) &> n - d_{\min}(\mathcal{C}) \\ &\equiv \\ d_h(\widetilde{m}, I_{\mathcal{C}}^{-1}(c_n)) &\leq \lfloor \frac{d_{\min}-1}{2} \rfloor\end{aligned} \tag{4.6}$$

It is impossible that two distinct neurons c_1 and c_2 satisfy Equation (4.6). By contradiction, it would mean that there exist c_1 and c_2 such that

$$d(c_1, \widetilde{m}) + d(c_2, \widetilde{m}) \leq 2\lfloor \frac{d_{\min}-1}{2} \rfloor < d_{\min}$$

The triangle inequality leads to $d(c_1, c_2) < d_{\min}$ which contradicts the minimum Hamming distance definition. Therefore, a message $\widetilde{m}$ will correspond at most to one activated neuron among $\mathcal{C}_{\mathcal{N}}$. In such case, the output of the network will correspond to the ML decoding of $\widetilde{m}$. If no neuron switch activated, this means that the message $\widetilde{m}$ is distant from all codewords of at least $\lceil \frac{d_{\min}-1}{2} \rceil$.

It is thus possible to decode any code using this simple neural construction.

Figure 4.8 shows the neural decoder associated with the $\mathcal{C}_H(2)$ code, which is an ML decoder. This decoder is also an encoder since it can produce the codeword associated with the provided, possibly partial, pieces of information.[4]

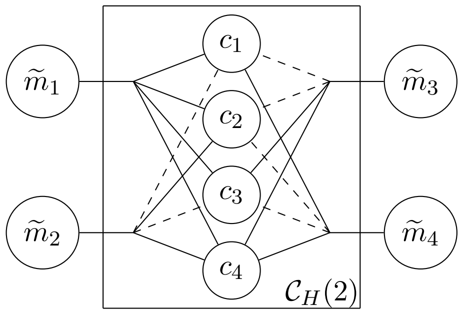


Figure 4.8: ML neural coder/decoder of the $\mathcal{C}_H(2)$ code. The message $\widetilde{m}$ is decoded using codeword neurons. Dashed edges represent weight -1 whereas full lines represent weight 1. The thresholds defined for the codewords neurons are set to 2.

Figure 4.9 shows an example of a neural decoder with the neural network unfolded to illustrate the iteration process.

An immediate drawback of this method is the number of neurons it requires.

[4]This comes from the fact that this specific code is such that it is possible to retrieve two missing bits in any of its codewords.

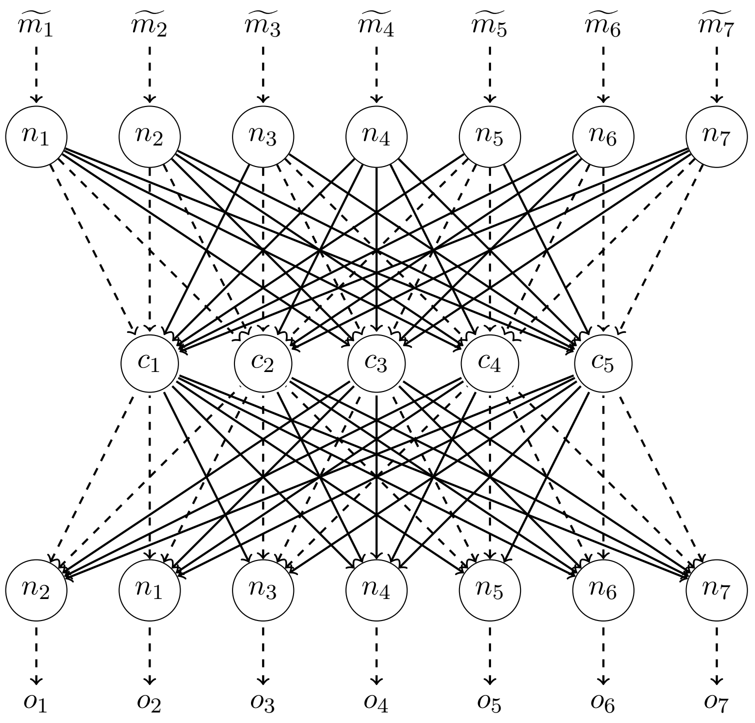


Figure 4.9: Example of a neural decoder associated with the code described in Figure 3.5. The network has been unfolded to illustrate the iterative process. Given an input message $\widetilde{m}$, it first projects this message to the input neurons $\mathcal{N}$. Then, it selects among the neurons in $\mathcal{C}_{\mathcal{N}}$ the only one corresponding to a codeword that is at less than a distance 1 to the message $\widetilde{m}$. Finally, it projects back its decision to the input neurons, giving the decoded answer (the vector $\mathbf{o}$).

This number grows linearly with both the dimension of the space n and the number of codewords. If the first one is rarely an issue, the number of codewords can grow exponentially with the dimension of the space (2^k). Moreover, this construction needs a prior study on the code to find its minimum distance. The question of how to implement the winner-take-all rule in neural networks has already been explored [33, 34] but the proposed solutions require a lot of connections. The next subsection presents an alternative that require a small number of added connections.

4.3.2 Maximum neural selector

4.3.2.1 Use of exponential neurons

Exponential neurons are among the various types of neurons found in the neurobiologic literature. An abstraction of these neurons behaviour is presented here:

$$f(v) = \begin{cases} \alpha\beta^v - \delta & \text{if } \alpha\beta^v - \delta \geq \gamma \\ 0 & \text{otherwise} \end{cases}$$

with $\beta, \alpha > 0$. γ is a threshold, α, β and δ are parameters that enable consideration of a large panel of exponential functions.

One can use the variations of the exponential function to let only the highest value expresses itself among the neurons in $\mathcal{C}_\mathcal{N}$. The iteration back to the input neurons will be processed as before. To illustrate this property, let us consider a bipartite neural decoder with the same architecture as in the previous section.

Rather than using an *a priori* knowledge about the decoder, let us consider that neurons have an exponential answer to their impulse. Considering the previous remarks and Equation (4.5), it is clear that the neuron in $\mathcal{C}_\mathcal{N}$ with the highest input is the one that should switch to active. Rather than preventing the other neurons from activating also, we will force the good one to have an activity large enough to mask all the other ones.

For instance, let us consider the code specified in Table 4.1. If the network is initialized with the received message 000001 then the initial scalar product obtained for each neuron in $\mathcal{C}_\mathcal{N}$, just before the exponential modification, will be 4 for the codeword 000000, 2 for two 010101 and 100011 and at most 0 for the others. Let us choose $\gamma = -1$, and $\alpha = 1$.

The correct codeword will be retrieved only if the activity of the neuron corresponding to the codeword 000000 is strictly larger than the sum of the activities of the other neurons in $\mathcal{C}_\mathcal{N}$ (we must consider the sum of those since it could counter-balance the decision made by the correct codeword). This leads to the condition that $\beta^4 > 2\beta^2$, that is $\beta > \sqrt{2}$.

This solution works in many cases, but still require to carefully choose the parameters. Moreover, it could lead to the consideration of neurons with astronomically large activity values, which hinders the biological plausibility.

4.3.2.2 Maximum selector with simple neurons

Another method for the selection of the maximal input among a population of neurons can be realized using the neuron model as described in Figure 2.3. This selection uses the next formula, which holds for any real numbers x and y [35,36]:

$$\boxed{\begin{aligned}\forall x, y, \max(x,y) &= \frac{x+y}{2} + \left|\frac{x-y}{2}\right| \\ &= \frac{x+y}{2} + \max\left(\frac{x-y}{2}, 0\right) + \max\left(\frac{y-x}{2}, 0\right)\end{aligned}} \qquad (4.7)$$

Therefore, a simple network can be associated with the formula to compute the maximum between two reals x and y (other solutions exist, see [37] for instance). This network is depicted in Figure 4.10.

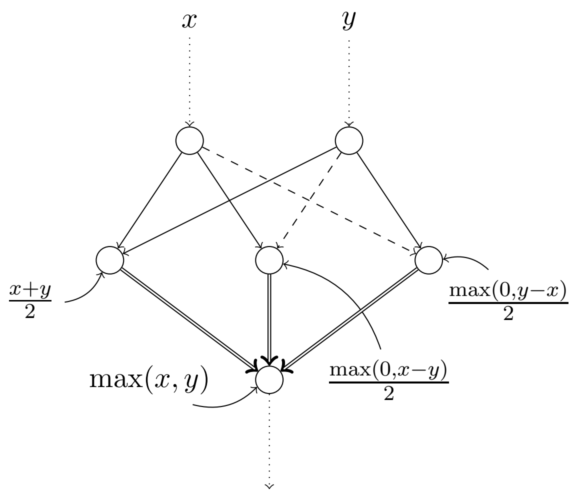


Figure 4.10: Maximum selector with neurons. Solid line edges represents weight $\frac{1}{2}$, dashed lines weight $-\frac{1}{2}$, and double lines weigth 1.

One can recursively use this construction to find the maximum in a set of n real numbers. This induction is depicted in Figure 4.11.

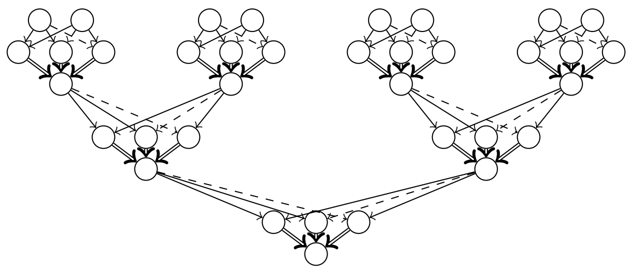


Figure 4.11: Maximum selector over 2^q (here: $q = 2$) values with neurons. Solid line edges represents weight $\frac{1}{2}$, dashed lines weight $-\frac{1}{2}$, and double lines weight 1.

When trying to decode the code described in Table 4.1, the exponential, maximum selection and (hard) ML decoders achieve exactly the same performance both in terms of MER or BER.

It is thus possible to decode any code using very simple neural networks but with the cost of a lot (linear number) of neurons.

4.4 Coded Hopfield networks

4.4.1 Hopfield networks and orthogonality

4.4.1.1 Orthogonal Hopfield networks

In some specific conditions, Hopfield neural networks with n neurons may learn up to n messages such that each one is correctly retrieved by the network if no error nor erasure is added.

This happens when input messages $\mathbf{d}^m$ are orthogonal:

$$\text{if } m_1 \neq m_2 \text{ then } < \mathbf{d}^{m_1}; \mathbf{d}^{m_2} >= 0$$

To understand this property, let us write again the dynamic equations of the model. If w_{ij} denotes the weight of the connection between neuron i and neuron j and after learning M messages $\mathbf{d}^m$, the following set of equations holds:

$$w_{ij} = \sum_{m=1}^{M} d_i^m d_j^m$$

Moreover, contrary to the initial model, we accept looping connections.

The decoding process at first iteration then takes into account all the contributions from the other neurons:

$$\begin{aligned} v_i^1 &= sign(\sum_{j=1}^{M} w_{ij} v_j^0) \\ &= sign(\sum_{m=1}^{M} d_i^m \sum_{j=1}^{M} d_j^m v_j^0) \end{aligned}$$

Thus, if the input to the neural network is a message $\mathbf{d}^\mu$, the value of neurons at step 1 is:

$$\begin{aligned} v_i^1 &= sign(\sum_{m=1}^{M} d_i^m \sum_{j=1}^{M} d_j^m d_j^\mu) \\ &= d_i^\mu \end{aligned} \quad (4.8)$$

This means that the network has already reached a fixed point. □

In other terms, a HNN made of n neurons may learn up to n orthonormal messages such that all of them are recognized by the network. Nevertheless, a very small perturbation on a word lead to catastrophic performance.

4.4.1.2 Bipartite Hopfield networks

One can use this specificity of the Hopfield neural network to increase the number of learned messages (we published this idea in [35]).

The idea is basically to associate orthogonal messages with message to learn such that the decoding process will benefit from this added orthogonality. To realize this association, we will consider a bipartite Hopfield neural network, which is a Hopfield network on a bipartite graph. Figure 4.12 depicts this structure.

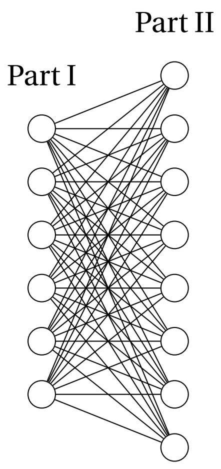


Figure 4.12: Model for the bipartite Hopfield network. In this example, the network is made of a part I with 6 neurons and a part II with 8 neurons. It contains a total of 48 connections.

As a direct consequence, the number of connections is greatly reduced. This property brings to the Hopfield model a simplified dynamic divided into two movements: neurons are updated consecutively, one part after the other.

Let us consider that part I contains k neurons indexed by i and part II contains L neurons indexed by j. In this network, one can learn messages on part I, completing part II by orthogonal messages that will be used for the retrieving process.

If part II is provided with correct information, the dynamic process will project the correct pattern back to part I. This property is directly given by Equation (4.8).

However, one still has to obtain the correct pieces of information on part II. The best way to retrieve information from part of it is to use a code designed for it. For instance, Hadamard codes can be used, as they are orthogonal. The next subsection presents a model that inserts such a code into the Hopfield bipartite model to increase its performance.

4.4.1.3 Orthogonal codes

Let us now consider orthogonal messages generated thanks to a certain code. During the learning phase, these codewords are associated one to one with input messages using a bipartite HNN.

In order to maximize the probability of success during the retrieving phase, part II will use a decoder. And as we are considering associative memories, this decoder will be a neural one. The dynamic of the network is thus updated to take into account this new device. Figure 4.13 depicts this new dynamic.

$$\begin{tikzcd}
\text{start} \arrow[r] & \text{Update of part II} \arrow[dl] & \\
\text{Neural decoding of part II} \arrow[rr] & & \text{Update of part I} \arrow[ul]
\end{tikzcd}$$

Figure 4.13: The dynamic of the bipartite coded Hopfield network.

The performance of the system depends mainly on two aspects: orthogonality and minimum distance of the code. Hadamard codes provide both of them perfectly.

The network learning process is as follows. When a new message is presented to part I of the network, a fresh unused codeword is presented to part II such that both of them are learned together as if they were forming a unique larger message.

Figure 4.14 shows a comparison of a classical HNN with $n = 150$ neurons where 25 messages have been learned with a bipartite coded HNN as previously described with 40 input neurons facing 256 code neurons where 256 messages have been learned. This is a fair comparison as the amount of stored connections in both cases is similar.

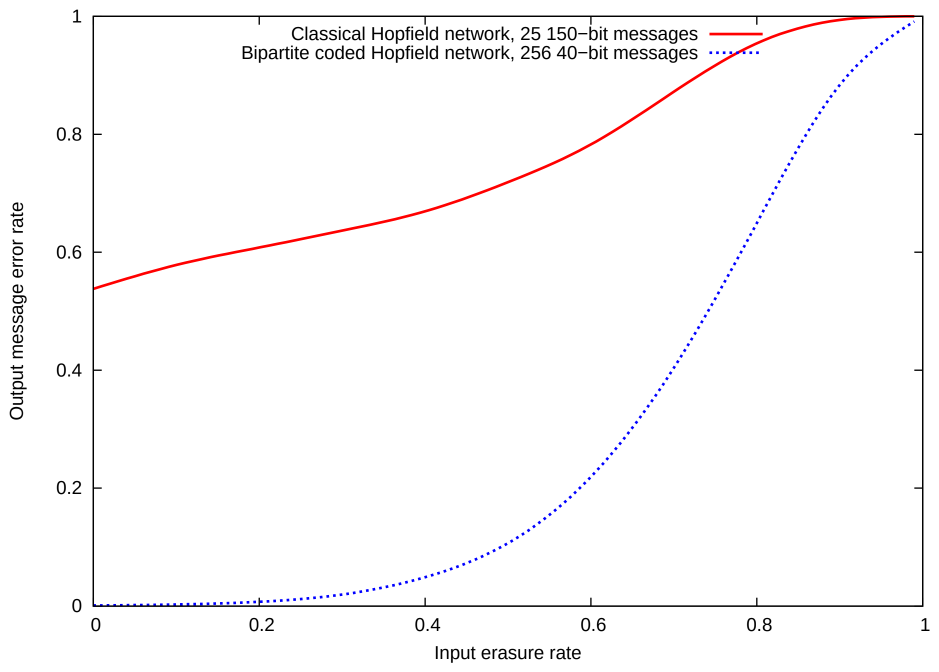


Figure 4.14: Comparison of performance between a classical HNN and a bipartite coded network with the same number of connections.

This result shows the considerable increase in performance coming from the use of a code in the Hopfield network. The number of learned messages (diversity) has increased a lot. Even the total amount of information learnt (capacity) has increased by a factor of 2.7.

Nevertheless, the maximum number of messages such a network can learn is bounded by the size of the code on part II. Moreover orthogonality restrains this number to the number of neurons. This use of orthogonal codes is therefore just a first step in the seek for better associative memories.

4.4.2 Projection onto large codes

4.4.2.1 Model

To transcend this limitation, a more subtle analysis is necessary.

Actually, the network failings in retrieving the learned messages is due only to the first step of the iterative decoding process. If the decoded codeword is correct on part II, the associated part I message will also be correctly retrieved.

Thus, in order to enhance performance one would try to limit this loss of information during the first step. A possible solution is to associate orthogonal codewords with other orthogonal codewords. Nevertheless, a direct confrontation between two codewords would not allow more messages to be learned. Thereby two other transformations are added to the network: first part I is split into several sub-parts, each one associated with a perfectly orthogonal code as presented before. Then those codes are put face to face with a larger one, with a lower constraint on orthogonality.

This way, both the perfection of the orthogonal coded Hopfield networks are kept locally and the error correcting ability of a large code is maintained on part II.

Figure 4.15 depicts this new model: messages are split into c sub-messages addressing $\kappa = \frac{k}{c}$ bits, each one connected locally to a bipartite coded Hopfield network with a Hadamard code. Those codes are then connected via another Hopfield network to another code through a larger network. If the orthogonal codewords are locally successfully retrieved, the entire message will also be correctly retrieved.

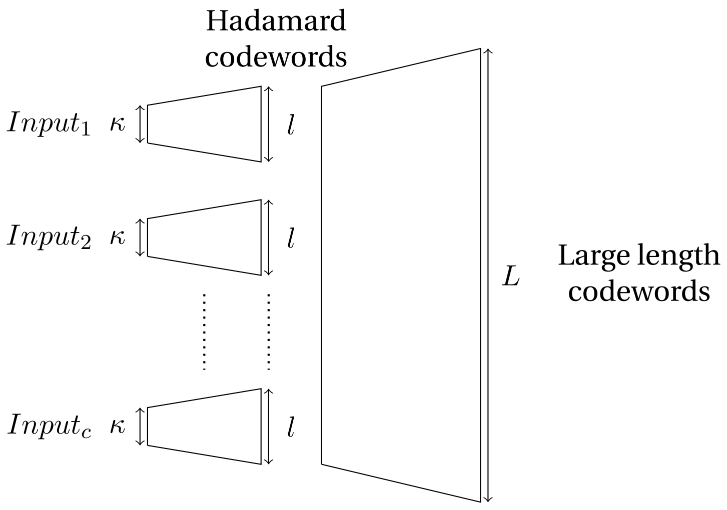


Figure 4.15: A way to associate a message composed of sub-messages of length κ with a large code through local orthogonal (Hadamard) codes of length $l = 2^{\kappa}$.

The large code, containing L neurons, is not necessarily orthogonal.

4.4.2.2 Performance

In the proposed model, pieces of information are stored in the connections of the network. Thus, a HNN to compare to would be one with a similar number of connections.

One could object that our network requires a lot more connections: those of the different neural decoders. Nevertheless, this number is insignificant in comparison to the number of connections in the HNN. Moreover, this material is independent of the set of learned messages and should thus not count in the amount of memory used.

On the other hand, a fair objection is that the number of learned messages being largely increased in our model, the connections of the coded Hopfield networks we are using are specified on more values than the classical HNN and therefore require a larger amount of information used. Yet this increase is no more than logarithmic with the number of learnt messages.

Figure 4.16 depicts the performance of the proposed model and the HNN with a similar number of connections and as a function of the number of learned messages.

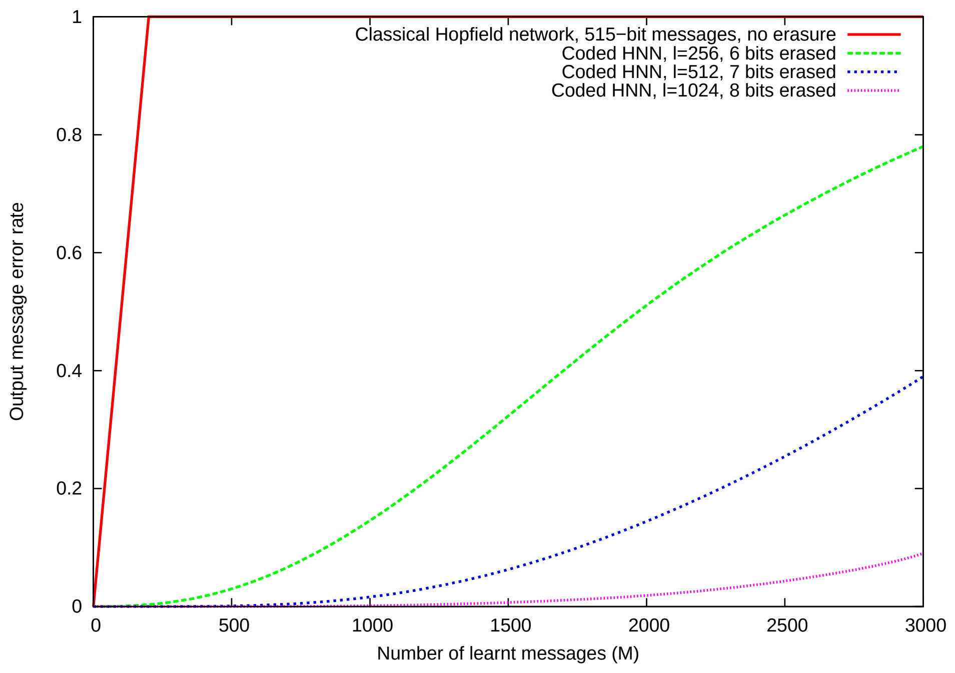


Figure 4.16: Compared performance of the proposed model with three different sizes for the orthogonal code ($L = 256$, $L = 512$, $L = 1024$) with a HNN with 515 neurons, containing roughly the same number of connections (than the one with $L = 512$). The large codes are random codes. Input messages, in the case of coded HNN networks, are made of $c = 4$ sub-messages of length $\kappa = 6$ bits each. For a targeted error rate of 10^{-1}, the coding gain in diversity is about 22, 45, and 75 for the three values of L.

Obviously, this is not a very surprising result as the network learns more but shorter messages.

Yet this model allows, with the same number of connections, to dramatically increase the number of learned messages. The capacity has also increased, but to a lesser extent.

In this chapter we have combined error correcting codes with neural networks. This allows us to considerably increase the number of learned messages in associative memories.

Two important codes have been introduced: constant weight codes and cliques. They are of prior interest as they can both be decoded in very simple neural networks. The next chapter explains how to combine these codes in order to increase once more the performance of associative memories.

5

Sparsity and networks of neural cliques

Contents

Sparsity is a very hot keyword in many domains, including associative memories [38–40]. Actually, codes and sparsity are not typically mixed as they are often incompatible: one has generally to choose between minimum distance or sparsity.

However, the clique code and the constant weight code are two codes that enable the considerations of sparsity. This chapter will show how one can combine them to produce a high performance associative memory.

Some improvements on the model are also presented: consideration of correlated messages and turbo-approach. Finally, a method to hold performance asymptotically is introduced.

5.1 Sparse networks

5.1.1 Sparsity benefits

In terms of information theory capacity, it is clear that a well-built device should be able to learn more messages if they are shorter.

This result can be mathematically expressed as follows. Let us suppose that the network is built on a symmetric graph with n neurons and no loops (as in a Hopfield neural network). If connections can take P different values, the maximum total amount of used information is:

$$S_{\max} = \frac{n(n-1)\log_2(P)}{2}$$

since there are $\frac{n(n-1)}{2}$ connections.

Now let us suppose that a network is able to achieve efficiency 1. The diversity, in the case messages are of length n, is thus:

$$M_{\max} = \frac{(n-1)\log_2(P)}{2}$$

This means that, considering P to be reasonable (*i.e.* a sub linear function of n), the network cannot expect to learn much more than a linear number of messages.

If one restrains messages to length k, this number becomes:

$$\boxed{M_{\max}^{sparse} = \frac{n(n-1)\log_2(P)}{2k}}$$

The bound has become quadratic with n.

Thus, the sparsity of messages, seen as the short length of messages compared to the number of neurons, brings a dramatic increase in the diversity of the network.

5.1.1.1 Sparsity in Hopfield networks

Hopfield networks are not directly suited to short messages representation. With no modification on the structure, one cannot learn such sparse messages and then retrieve them without adding additional information. This is easily understandable: the decoding process will give values to every bit as if they were all erased.

If it was possible to work on sub-regions of a Hopfield network, the diversity and capacity would greatly increase.

A possible alternative is to add to any retrieval process the information on the bits locations in the sparse message. This implies to add a character to denote the absence of a symbol and therefore to change the representation space to a sparse one. This idea seems interesting as the efficiency of the Hopfield network tends to 0 as n tends to infinity, a lot of smaller more effective networks could result in better performance.

Let us consider a classic Hopfield model with n neurons. Messages to learn are now of length $k < n$ and over a new alphabet $\mathcal{A} = \{-1; \epsilon; 1\}$ where ϵ denotes the absence of a character. Let us insist that $\epsilon \neq 0$ as it does not represent an erased bit that has to be retrieved but a bit that should not be considered during the decoding process. The learning process can be slightly modified: after the learning of M messages $\mathbf{d}^m$, W_{ij} - the weight of the connection between neurons i and j - is such that:

$$W_{ij} = \sum_{m \leq M} d_i^m \otimes d_j^m \tag{5.1}$$

where

$$a \otimes b = \begin{cases} 0 & \text{if } a = \epsilon \text{ or } b = \epsilon \\ ab & \text{otherwise} \end{cases} \tag{5.2}$$

In the retrieving process, the alphabet of input messages is enlarged to $\mathcal{A}_{in} = \{-1; 0; \epsilon; 1\}$. During the decoding process, only the neurons with a value different from ϵ will be updated.

There are several ways to implement this new representation in order to increase diversity.

A first one consists of splitting the network in several totally independent sub-networks. It is then possible to theoretically express the gain of such a network as follows: the diversity of the HNN is close to $M_{\max}(n) = \frac{n}{2\log(n)}$. If one splits the network into $\frac{n}{k}$ parts of k neurons each, those sections will be able to learn each up to $M_{\max}(k) = \frac{k}{2\log(k)}$ messages each, which leads to a total diversity of:

$$M_{parallel} = \frac{n}{k}\frac{k}{2\log(k)} = \frac{n}{2\log(k)} \tag{5.3}$$

if k is large enough.

Thus, in comparison to Equation (2.3), it is possible to increase the number of message simply by considering parallel and independent networks.

In this perspective, a lot of connections are unused (all those between distinct sub-networks). Another solution would be to learn messages of length k anywhere on the network. The performance will be similar to that of a parallel HNN if all connections have been used, on average, a comparable number of times - that is $\frac{k}{2\log(k)}$. There are $\frac{n(n-1)}{2}$ connections, and each time a new message of length k is learned, $\frac{k(k-1)}{2}$ are used. The number of messages one can learn - considering they are perfectly uniformly distributed over the network - is therefore:

$$\boxed{M_{sparse} = \frac{n(n-1)}{2}\frac{k}{2\log(k)}\frac{2}{k(k-1)} = \frac{n(n-1)}{2(k-1)\log(k)}} \tag{5.3}$$

leading to a quadratic number of learned messages. This equation holds if k is large enough.

Figure 5.1 depicts the performance of this sparse HNN compared to the classical one and the parallel one in terms of diversity and for the same amount of neurons. It insists on the dramatic increase of diversity due to the use of sparsity.

Figure 5.2 depicts the performance of the sparse HNN compared with the parallel HNN and the classical HNN in terms of capacity and for the same amount of information stored. This figure shows that besides its interest in terms of diversity, sparsity may also produce important gains in capacity. Note also that the results are identical for parallel and sparse networks since the former offers a lower diversity but also use much less connections.

Nonetheless, these considerations require modification of the input alphabet. The next sections show how associative memories can benefit from this sparsity consideration without any need for modifying the learning space.

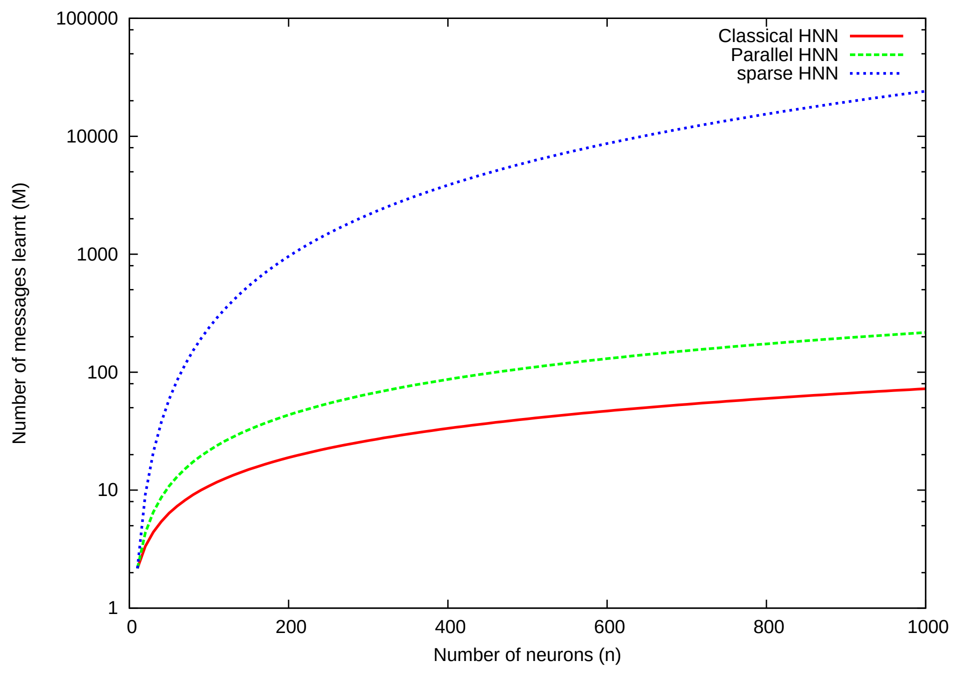


Figure 5.1: Comparison of the diversity of the sparse HNN with the parallel HNN and the classical HNN with the same error probability and as a function of the number of neurons.

5.1.2 Network sparsity

5.1.2.1 Minimum distance versus sparsity

The idea presented in this section can be seen as the natural continuation of many models presented thus far.

The simplest way to “sparsify” information is to associate it with a sparse code, for instance the constant weight code with $w = 1$.

Figure 5.3 represents such an association where input messages are coded over binary values ($\{-1; 1\}$). Each possible sub-message is then associated one to one with a neuron.

It is interesting to note that such a transformation can be obtained by inverting the neural network decoders introduced in Section 4.3. Actually, these decoders associate a unique neuron with each codeword.

The idea is thus to consider this neuron as the index of a codeword rather than a representation of its content.

Those codewords can then be associated as in the coded Hopfield network model.

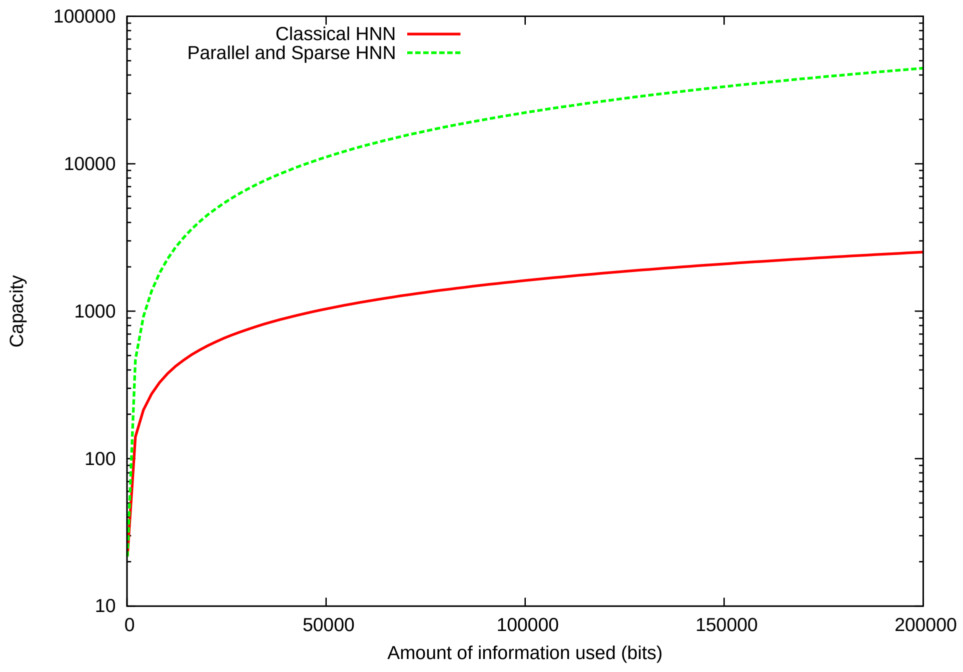


Figure 5.2: Comparison of the capacity of the sparse HNN with the parallel HNN and the classical HNN with the same error probability and as a function of the total amount of information stored.

Note that this model can also be obtained from the bipartite structure introduced in Section 3.3 where a coding rule has been added to the way cliques are built.

5.1.2.2 Getting rid of the large code

The large code introduced in Section 4.4 was necessary as the amount of learned information largely overpass the bound of classic Hopfield networks. Thus connections were used a large number of times, decreasing significantly their discriminative abilities.

Because of the structure of the sparse code, it is now possible to get rid of the large code and of the HNN principles.

We published this construction in [36,41]. The messages to learn are divided into several sub-messages, each one addressing a specific cluster in the network. To each sub-message will actually correspond a unique neuron in each cluster, as in Figure 5.3. Those neurons are then all connected together, printing a clique into the network. As the process only affects a very small number of connections,

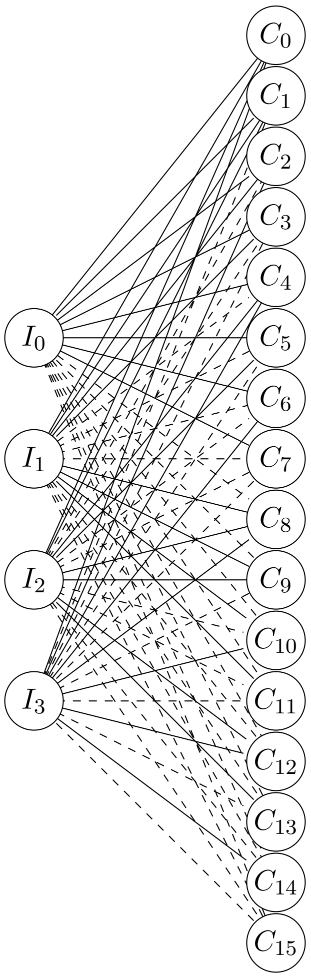


Figure 5.3: Example on how to associate a constant weight code with weight $w = 1$ with input messages using a neural network. Input bits are indexed from I_0 to I_3 and associated codewords from C_0 to C_{15}. In this network, neurons on the right are activated if their input is at least equal to three. An input with no erasures will correspond to the activation of a unique neuron on the right part. Note that this network is a specific case of the one depicted in Figure 4.7 with the restriction that every input corresponds to a unique associated codeword.

we can get rid of the connection weights. Therefore, the network becomes a binary one: a connection exists or not. Figure 5.4 depicts this new learning process.

So the learning of a message mixes two codes. First, messages are mapped to sparse representations using a constant weight code with weight one. Then obtained messages are learned through a clique in the network.

Obviously, a first importance parameter to assure a good categorization between learned and not learnt messages is the network density. A density close to 1 would lead to an over-loaded network, which cannot retrieve learnt messages.

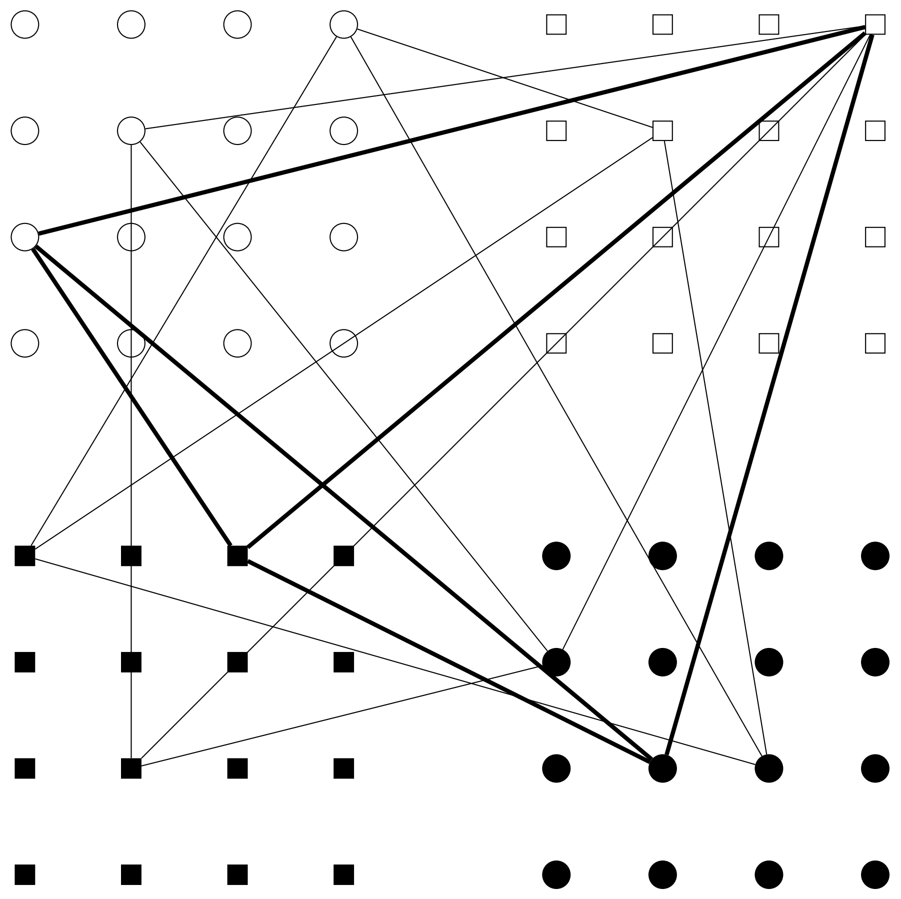


Figure 5.4: Learning process illustration. The pattern to learn (with thick edges) connects neurons from 4 clusters of 16 fanals each (filled circles, filled rectangles, rectangles and circles). It represents a geometric figure (in this case a tetrahedron) which is finally printed in the network.

Let us denote by c the number of clusters in the network and by l the number of neurons per cluster (there are a total of $n = cl$ neurons in the network). After the learning of M uniformly distributed random messages, connections in the network may be considered independent, which is not much of an approximation if $M \gg c$. The expected density d is then directly connected to M by the following formula:

$$\boxed{\begin{aligned} d &= 1 - \left(1 - \frac{1}{l^2}\right)^M \\ &\approx \frac{M}{l^2} \text{ when } M \ll l^2 \end{aligned}} \tag{5.2}$$

Figure 5.5 shows the evolution of the density d with the number M of random messages learned, for four values of l. Note that this density does not depend on the total number of neurons n nor on the number of clusters c but on the cluster size l, meaning that from this point of view, and given a total number of neurons, a small number of clusters is preferable. The choice of c may also depend on other criteria, such as the targeted retrieval error rate.

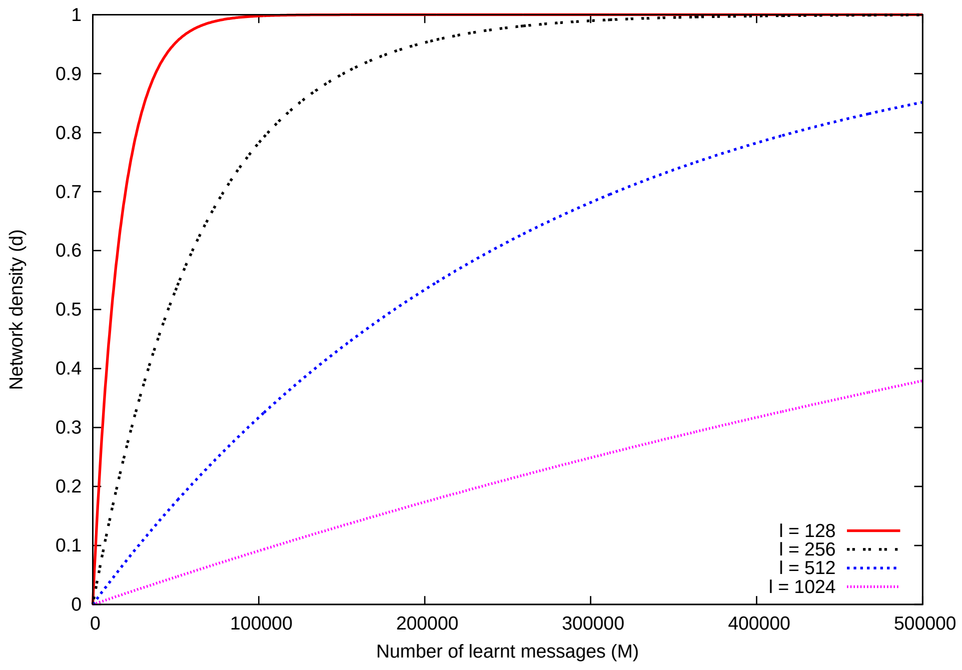


Figure 5.5: Evolution of the network density when learning random messages for 4 cluster sizes.

Note that the organization of the network in clusters has reduced the number of possible connections with respect to the complete graph ($\frac{(c-1)n^2}{2c}$ to be compared with $\frac{n(n-1)}{2}$) but this reduction, and therefore the reduction in available memory, is low and acceptable (25% for $c = 4$, for instance).

5.1.2.3 Decoding process

The decoding process is divided into two steps, global and local - that is one for each code.

Once a partially erased or erroneous message has been presented to the network and the associated neurons have been selected using networks as depicted in Figure 5.3, the decoding process iterates the two steps : first the global decoding, using cliques in the network, then the local one, using the winner-take-all rule. This way the neurons have binary values between each iteration.

The network stops once it reaches a stable state, that is a fixed point in the decoding process (usually 1 or 2 iterations are sufficient).

Note that input messages do not need to be binary. The only important aspect is to keep a unique neuron associated with each possible character. Nevertheless, the binary consideration allows a direct comparison with the Hopfield network.

We call the neurons associated to each sub-message fanals[1] as an input unaltered message will correspond to only one of them in each cluster. Also, it is more likely that fanals should be associated to populations of neurons in the brain (micro-columns for instance) rather than neurons.

Let us denote by n_{ij} $(i \leq c, j \leq l)$ the j-th fanal of the i-th cluster and by v the function that associates these neurons with their values. Then the global decoding can be written as:

$$\forall i, j, v(n_{ij}) \leftarrow \sum_{i'=1}^{c} \sum_{j'=1}^{l} w_{(i'j')(ij)} v(n_{i'j'}) + \gamma v(n_{ij}) \tag{5.3}$$

Let us point out that Equation (5.3) formalizes a message passing algorithm including a memory effect, with parameter γ. The memory effect is necessary, as it prevents the network from rejecting a learned message. On the other hand, its value may have a very important influence on the performance of the network: if it is too large, the network will not be able to correct errors. If it is too low, a correct decision may be counter-balanced during the decoding process.

Actually, the memory effect, if equal to 1, presents the same behaviour as if the graph had loops on all its neurons. We consider that it is preferable to use the former since loops are not biologically plausible. Moreover, loops offer less control on the activity of the network.

The local decoding process in the i-th cluster is then described by the following algorithm:

$$v_{i,\max} \leftarrow \max_{j} v(n_{ij}) \tag{5.4}$$

$$\forall j, v(n_{ij}) \leftarrow \begin{cases} 1 & \text{if } v(n_{ij}) = v_{i,\max} \\ & \text{and } v_{i,\max} \geq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

This algorithm formalizes a "winner-take-all" process. Nevertheless, (5.5) allows several fanals to switch on simultaneously in the case they achieve the same score. This property is very important as it puts on hold ambiguous characters while others are being retrieved without ambiguity. The process will hopefully

[1]Fanals are beacons in the sea that help guide boats. They are often the only light source in the night.

later be able to make a unique decision.

Step (5.5) uses a threshold σ to control the activity of this local decoder if needed. This threshold depends on the application: in classification the largest possible value leads to the best results whereas in associative memories it benefits from being more carefully chosen. In the latest part of this document, the threshold will also be used to prevent an epileptic propagation of the signals towards the entire network.

Thus, an iteration in the decoding process can be written as:

$$\left[\begin{array}{c} \text{Use the global decoding} \\ \forall i, \text{ use the local decoding on cluster } i \end{array}\right. \tag{5.6}$$

5.1.3 Performance

5.1.3.1 As a classifier

Neural networks have for long been studied in the field of classification.

Among the different applications aimed by classification, one is to generalize some learning sets in order to automatically classify unknown inputs, as in *OCR*[2] for instance. It is clear that the proposed network could be used as such a classifier, yet these considerations have not been studied.

Another branch of classification is interested in set implementations, which can be seen as a degenerated form of the previously described application where learning sets are exhaustive, like in intrusion detection systems for instance. This document considers only this latter aspect.

More formally, let us denote E the set of messages to learn. A perfect classifier, as we consider them, is thus a device that recognizes all the messages (and only them) in E.

If we denote by F the set of learned messages - that is the messages that have been actually learnt-, a measure of performance over the network is a measure of differences between sets E and F. Equation (5.3) implies $E \subseteq F$, so the probability that a learnt message is not recognized, called the error probability of the first kind[3], is zero. Hence the only possible errors remaining are of the second kind[4]: unlearnt messages accepted by the network.

In the sequel two measures are presented. The first one is the probability $P(x \in F | x \notin E)$, that is the error probability of the second kind. The second one considers the size of F relatively to that of E. As a matter of fact, even with a good

[2] Optical Character Recognition.
[3] Or Type I error.
[4] Or Type II error.

error probability of the second kind, there may be a lot of unlearned recognized messages in an exhaustive test.

The first measure $P(x \in F | x \notin E)$ is close to $P(x \in F)$ as E is supposed to be of reasonable size ($E \ll 2^k$). Given a threshold $\sigma = c$ coupled with $\gamma = 1$, this latter probability can be easily estimated as follows. Indeed, an input on the network will remain unchanged only if all the connections between its associated fanals exist, that is only if the corresponding clique exists. Considering again that connections are independent, as was the hypothesis to establish Equation (5.4), this leads to the following formula:

$$P(x \in F) \approx d^{\frac{c(c-1)}{2}} \tag{5.7}$$

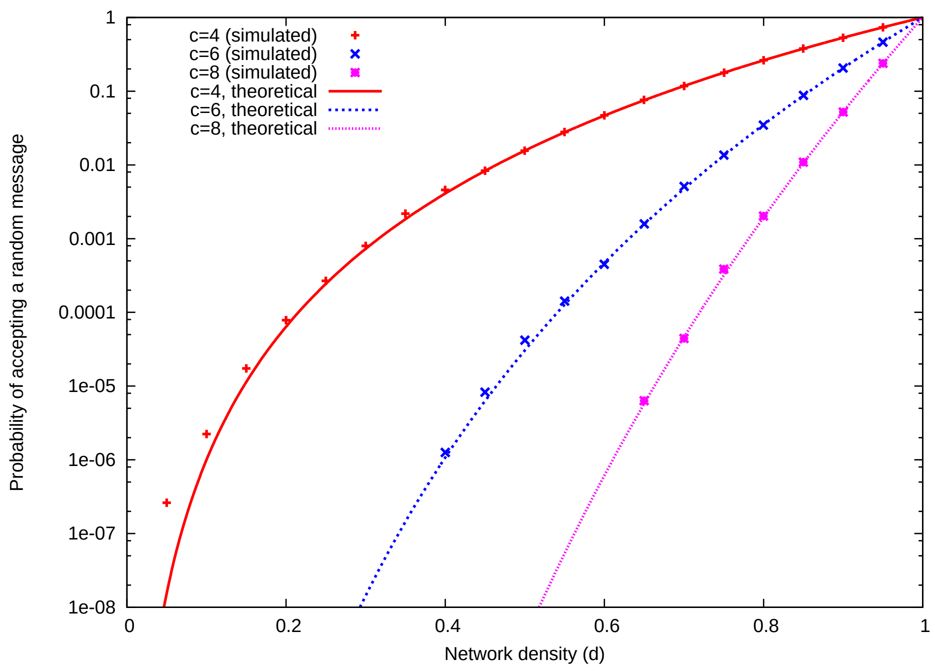


Figure 5.6: Probability to accept a random message for 3 numbers of clusters with size $l = 512$ and for $\sigma = c$. Both simulated points and theoretical curves are represented.

Figure 5.6 shows the evolution of this probability for three numbers of clusters with size $l = 512$ and for $\sigma = c$. It is surprising to remark that even with a very important density (up to 80% for $c = 8$), the performance remains very good (around 10^{-2} in error probability). Coupled with Figure 5.5, it shows that a network with $n = 4096$ neurons and $c = 8$ clusters may learn more than 400000 messages of length 72 with still very good discrimination ability.

Nevertheless, as pointed out before, an error probability of 10^{-2} corresponds to a lot of unlearned messages possibly accepted by the network, and that could be more numerous in an exhaustive test than the number of messages learnt. This number of accepted messages can be estimated from Equation (5.7) this way:

$$\#F = 2^k P(x \in F) \approx 2^k d^{\frac{c(c-1)}{2}} \tag{5.8}$$

A second measure of performance is to compare the size of F relative to that of E. Figure 5.7 draws the evolution of the ratio $\frac{\#(F-E)}{\#E}$ for three numbers of clusters with two sizes: $l = 256$ and $l = 512$.

Contrary to the previous consideration, this figure shows that this ratio is rapidly bad for too large a density. Also one may note that increasing the size of clusters does not improve performance as the number of possible messages is considerably enlarged. For instance, changing $l = 256$ into $l = 512$ increases this number by a factor of 256 in the case of $c = 8$. Meanwhile the number of learned messages has only grown by a factor of four (for the same density). A good performance considering a set implementation would be to restrict density to 0.15 approximatively.

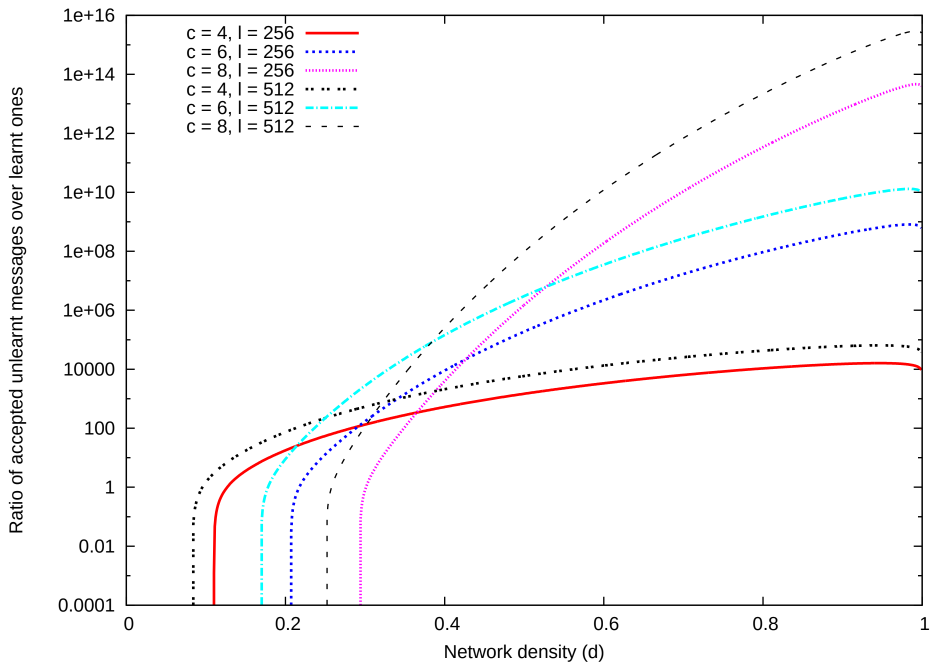


Figure 5.7: Ratio of the number of unlearned accepted messages over the learnt ones as a function of the network density and for 3 different number of clusters of size $l = 256$ and $l = 512$.

Table 5.1 shows the comparison of performance obtained with the same amount of memory used in an HNN and in the proposed network for $c = 4$ and $l = 512$.

The material needed for the local decoders is not taken into account as it does not depend on the learning sets. Those results show the very good performance of the proposed networks to achieve a go/no-go sort. Note also that the efficiency of the proposed network, in this experiment, exceeds 100%, which should not be surprising since messages are not ordered (see 2.4.3).

Model	HNN	Proposed network	ratio
Memory used (bits)	1.6×10^6	1.6×10^6	1
n	740	2048	$\times 2.8$
Message length	740	36	$\div 21$
First kind error probability	9%	0%	$\div \infty$
Second kind error probability	almost 0	almost 0	≈ 1
Diversity	56	60000	$\times 1071$
Capacity	4.1×10^4	2.2×10^6	$\times 52$
Efficiency	2.6%	137%	$\times 52$

Table 5.1: Comparison of performance between the HNN and the proposed network with $c = 4$ and $l = 512$ for the same amount of memory used, in the case of a go/no-go sort application.

Compared to the estimated bound on efficiency depicted in Figure 2.6, which is around 138%*, this shows the remarkable performance in terms of efficiency of the proposed network.*

5.1.3.2 As an associative memory

To be fairly compared with the Hopfield network which was originally introduced as such, a second comparison is to consider the performance of the proposed network as an associative memory.

The network is thus provided with partial information, some clusters receiving no information at all. The network can then try to retrieve the missing data. Contrary to the classification problem, error probability of the first kind is no longer zero as clusters with no provided information may induce an ambiguous decision on others. We will now consider the threshold σ to be zero.

The error probabilities, in the case of a single iteration, can once again be estimated by very simple equations. However, unlike the classification problem, iterations can significantly increase performance (see Figure 5.9).

After a single iteration and when only one cluster is not provided with information, the probability of electing the correct erased fanal is given by the following equation:

$$P_{retrieve} = \left(1 - d^{c-1}\right)^{l-1} \tag{5.9}$$

The probability that the other clusters decisions is not counter-balanced by the ambiguity produced by the one with no information is:

$$P_{remain} = \begin{cases} \left(\left(1 - d^{c-2}\right)^{l-1}\right)^{c-1} & \text{if} \quad \gamma = 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.10}$$

Considering that the memory effect is actually used ($\gamma > 0$), the error probability of recovering the message is:

$$P_e = 1 - P_{retrieve} = 1 - \left(1 - d^{c-1}\right)^{l-1} \tag{5.11}$$

Given (5.4), this probability is such that:

$$P_e = 1 - \left(1 - \left[1 - \left(1 - \frac{1}{l^2}\right)^M\right]^{c-1}\right)^{l-1} \tag{5.12}$$

This equation can be generalized as follows: if the number of clusters c_e without provided information is larger than one, the error probability becomes:

$$P_e = 1 - \left(1 - \left[1 - \left(1 - \frac{1}{l^2}\right)^M\right]^{c-c_e}\right)^{(l-1)c_e} \tag{5.13}$$

It is interesting to point out that the error probability decreases when c increases. On the other hand, l has two impacts on it: the first one is the factor $\frac{1}{l^2}$ which is the most important parameter: large clusters lead to better performance. The second one is the power parameter, which has a negative impact on performance. Note that l has finally a positive impact on the performance of the network but the asymptotic efficiency tends to zero as l tends to infinity, as in the Hopfield network. The next section will introduce another level of sparsity to avoid this drawback.

When the number of messages tends to zero, and for a reasonable cluster size: $l \gg 1$, this probability is close to:

$$P_e \approx l c_e \left[\frac{M}{l^2}\right]^{c-c_e} \tag{5.14}$$

Figure 5.8 draws the evolution of the message retrieval error rate when one of the four clusters of size 512 is not provided with information, as a function of the number of learned messages. This figure also draws the theoretical curve from (5.13).

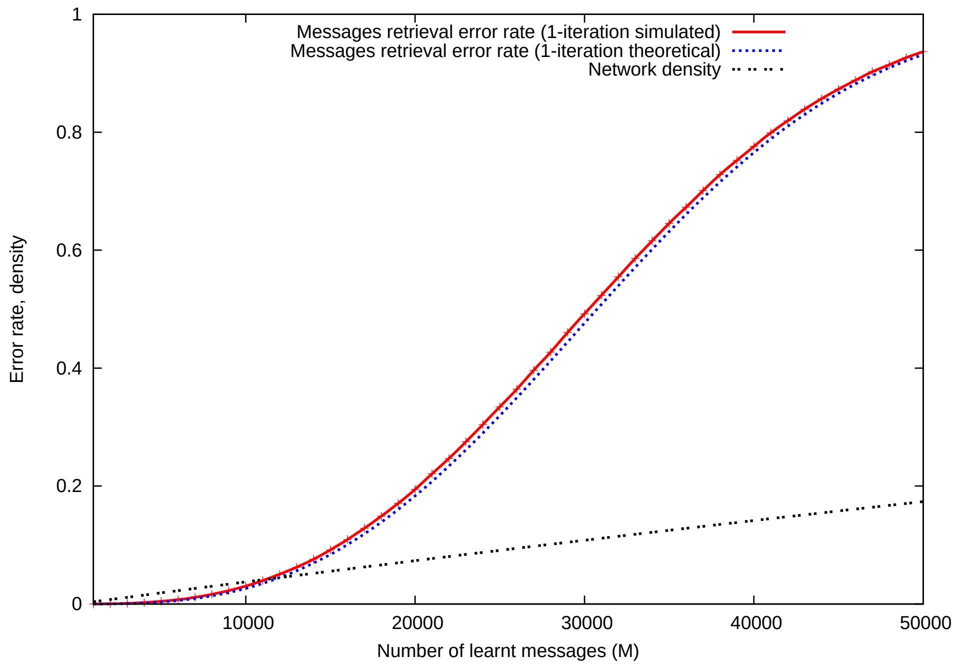


Figure 5.8: Evolution of the error rate when retrieving a learned message with 1 cluster with no provided information in a network with 4 clusters of size 512 as a function of the number of learnt messages. The simulated and theoretical curves as well as the network density are represented.

The optimal number of clusters, given a targeted error probability P_0, a number of neurons n and a proportion of clusters with no input information of $\frac{1}{2}$, can be easily obtained from (5.14) as follows:

$$c_{opt} = \log\left(\frac{n}{2P_0}\right) \tag{5.15}$$

For instance, the approximated optimal number of clusters for a targeted error probability $P_0 = 0.25$ with $n = 2048$ neurons is 8. Figure 5.9 draws the evolution of the error rate in message retrieval when half the clusters of such a network with $c = 8$ have no information, as a function of the number of learned messages and after four iterations. The theoretical curve for a single iteration from (5.13) is also drawn, showing the interest of iterative process in this situation.

The simulation shows that such a network of 2048 *neurons can learn up to* 15000 *messages of* 64 *bits each and retrieve them with a very high probability even when they are erased up to a half. This is, to our knowledge, unprecedented performance.*

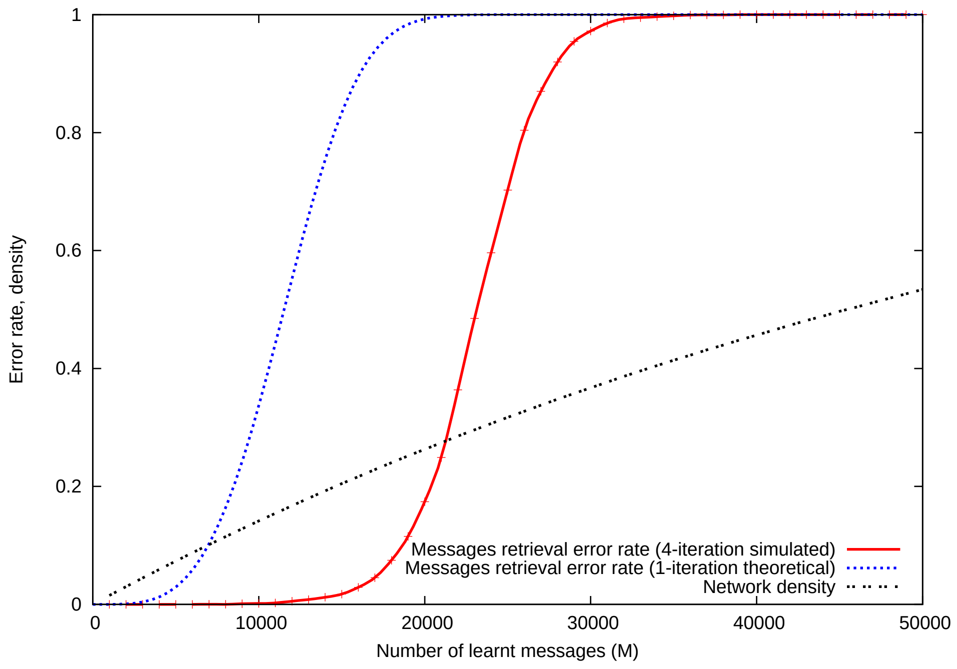


Figure 5.9: Evolution of the error rate when retrieving a learned message after 4 iterations with 4 clusters having no information in a network with 8 clusters of size 256, as a function of the number of learnt messages. The theoretical curve for a single iteration and the network density are also drawn.

Once again, the performance obtained with the model proposed in this paper is dramatically better than that obtained by the HNN. Table 5.2 compares the two models for the same amount of memory used and half the input erased.

Figure 5.10 depicts the gain of capacity from the HNN to the proposed network. The given curve considers a network with $c = 8$ clusters where one is not provided with information and the error probability in the retrieving process is close to 10^{-2}. This latter condition is severe as the equivalent HNN presents worse error probabilities even without erased inputs. The figure includes also the theoretical curve for a hypothetic network with efficiency equal to one. The proximity between this latter curve and that of the proposed network is worthy of note.

Model	HNN	Proposed network	ratio
Memory used (bits)	1.8×10^6	1.8×10^6	1
n	790	2048	$\times 2.6$
Message length	790	64	$\div 12$
Error probability	9%	2%	$\div 4.5$
Diversity	60	15000	$\times 250$
Capacity	4.7×10^4	9.6×10^5	$\times 20$
Efficiency	2.6%	52%	$\times 20$

Table 5.2: Comparison of performance between the HNN and the proposed model with $c = 8$ and $l = 256$ for the same amount of memory used, when both are used as associative memories.

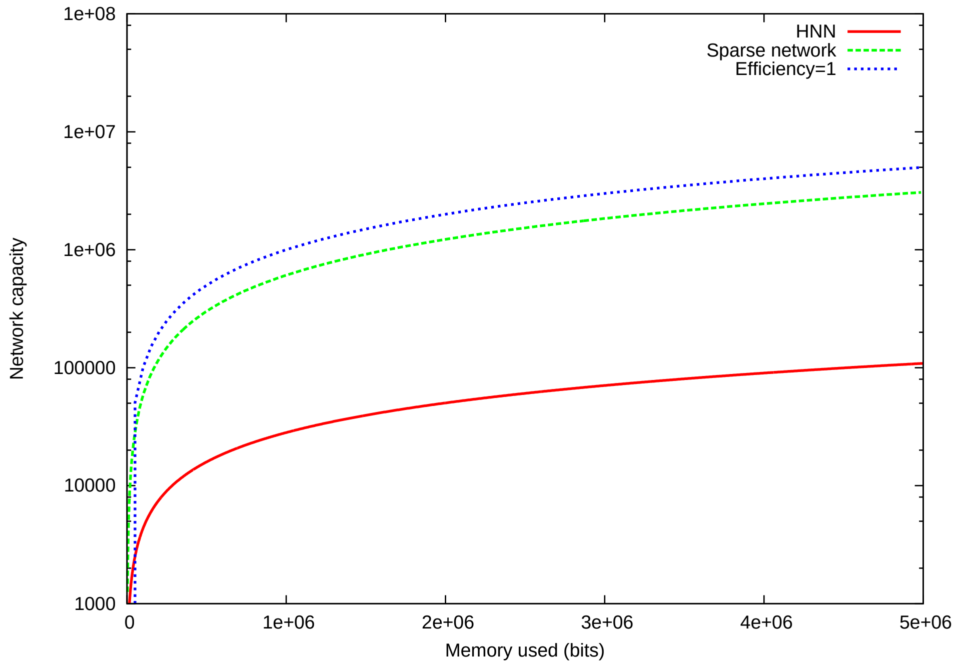


Figure 5.10: Capacity of the proposed network compared with that of HNN as a function of the amount of memory used, in the case of $c = 8$ clusters and a targetted error probability of 0.01 when one is not provided with information. The theoretical curve corresponding to efficiency equal to 1 is also provided.

5.1.3.3 Resilience

One of the prime interests of this network with regards to the Hopfield networks (and the other versions such as the Boltzmann machine [20]) and biological plausibility is its resilience.

It is certain that if we add noise to connections, it is likely that performance of the overall network will decrease. However, simulations show that this loss is acceptable. Consider for instance Figure 5.11 where connexions have been modified using noise with standard deviation 0.37.

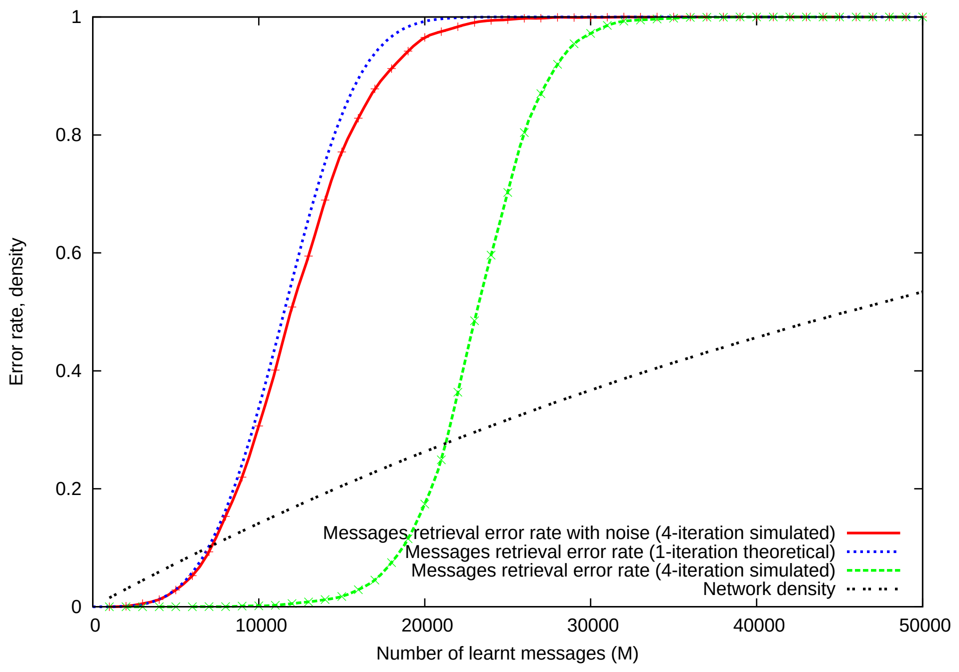


Figure 5.11: Comparison of performance between the sparse network with and without noise on connections as a function of the number of learned messages.

This figure shows that the performance of the network remains acceptable even when considering very significant noise added to connections. An equivalent noise on a HNN would have lead to catastrophic performance.

5.2 Turbo sparse networks

5.2.1 Correlated messages

The performance results obtained are valid while making the assumption that learned messages are i.i.d.. We show in this section that very good performance can be achieved even if the learning set has significant correlation.

The proposed network mechanics is based on the creation of a particular correlation materialized by cliques in the network. Nevertheless, the correlation coming from the learning set can correspond to worse performance of the network.

More precisely, one can distinguish two types of correlation. The first one is intrinsically bound to the learning set and therefore cannot be counterbalanced. For instance, if the learning set contains both messages “brain” and “train”, and if one of both is presented to the network with no information on the first character, this latter will face an ambiguity in retrieving the initial message. In such a case, the decision of any device could only be arbitrary.

On the other hand, the correlation of the learning set may create artificial ambiguities on the proposed network. Consider for instance the learning of the messages “brain”, “grade” and “gamin”. If all characters are mapped one to one to a cluster, the network will also contain the clique associated with the message “grain” which was not learned. Figure 5.12 depicts this correlation issue.

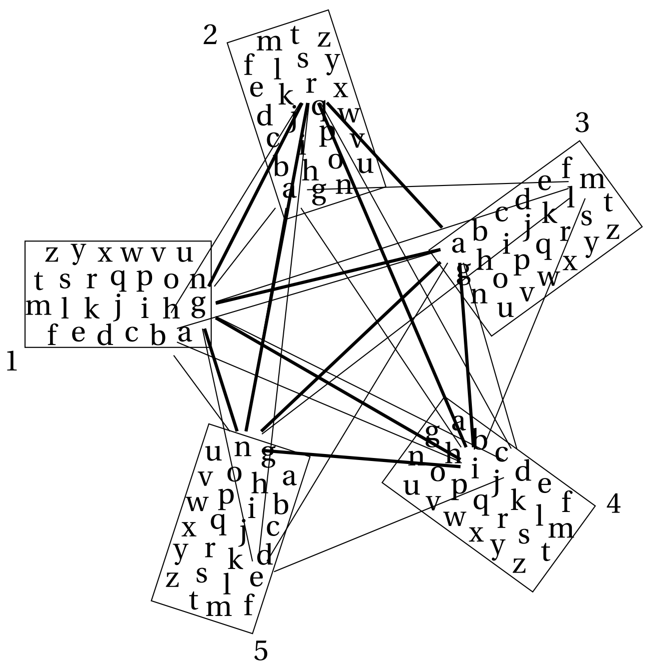


Figure 5.12: Example of a sparse network after the learning of the messages “brain”, “grade” and “gamin”. During this process, the clique (represented with thick connections) corresponding to the message “grain” has also been added to the network despite the fact it has not been learned.

Performing the learning on a set with highly correlated messages, this particular event is likely to happen often. In order to avoid this side effect, a simple solution is to add to messages to learn a context, that is a hidden signature.

Consider for instance a network made of ten clusters, each one made of twenty-six fanals - one for each letter in the Latin alphabet. In this network are learned all the English words made of five letters. Input messages are therefore only addressing five of the ten clusters. During the learning process, the five unmapped clusters are initialized to a random value. For instance, if we consider the same example as before with words “brain”, “grade” and “gamin”, a clique will also be created between the five clusters associated with the characters in input messages but probably not between the other ones. Figure 5.13 depicts this idea with a single hidden signature.

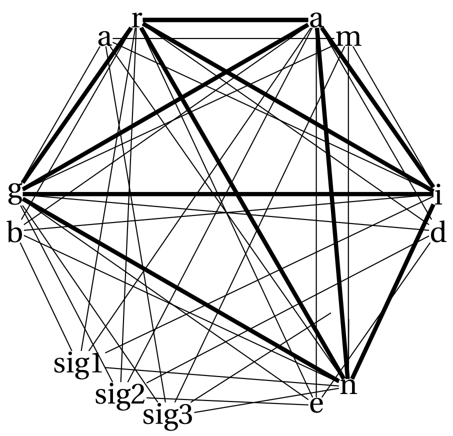


Figure 5.13: Example of a situation where words "brain", "grade" and "gamin" have been learned with a single hidden signature. The word "grain" is therefore not recognized by the network as it does not correspond to a clique (corresponding connections are thick): no signature is associated with every letter in the word. The probability to avoid the creation of such unwanted cliques in the network grows with the number of added signatures.

5.2.2 Turbo approach

In order to enhance performance on correlated messages, a turbo approach can also be performed.

Let us consider for instance that we aim at learning all the French words made of six characters, that is about 16000 words according to our dictionary. Considering the equations of Section 5.1.3, using clusters of size $l = 26$ (or slightly more with accents and special characters) - corresponding to the natural use of characters to split words - would result in an overfilled network with a very low probability of success both in classification or associative memory.

If characters are grouped into pairs instead, the density will decrease down to 3.4%. However, the number of clusters is reduced to three and therefore offers a relatively low redundancy gain.

One can artificially increase the number of clusters and the robustness of the learning process by duplicating some pieces of information. This added diversity will hopefully help strengthen the reliability of the decoding process. Figure 5.14 depicts such a way to transform characters in words of length six into six pairs of partially redundant symbols, materializing a turbo-approach.

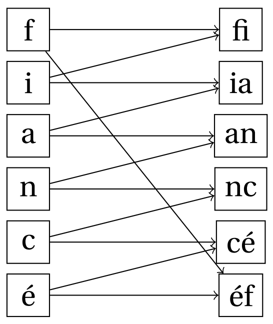


Figure 5.14: Example of a way to transform characters to partially redundant symbols allowing a message-passing approach in the decoding process. For instance, if characters "f" and "i" are erased, then the symbol "fi" will be completely erased and symbols "ia" and "éf" will be partially erased. If "f" and "a" are erased in the input, then symbols "éf", "fi", "ia" and "an" will be partially erased. Note that, as we share the erasures among several symbols, we are likely to benefit from iterations.

Using the construction depicted in Figure 5.14 plus six random signatures as described in the previous section, performance in retrieving a French word among the 16000 *learned ones when two characters are erased increased from a probability* $p = 30\%$ *of success to approximatively* $p = 80\%$. *Moreover, the latter corresponds to the limitation due to the intrinsic correlation of the learning set itself. The number of neurons in the network is 8112 (12 clusters of 676 neurons each).*

5.3 Towards a fourth level of sparsity

Despite very interesting performance, the proposed network suffers from a limitation: in associative memory applications its performance is mainly function of the size of clusters l and not their number c. As a matter of fact, the length of learned messages grows linearly with c, and logarithmically with l. If one wants to learn many messages, it is thus preferable to increase l rather than c.

This means actually two things: first the size of learned messages is strongly dependent on the size on the networks. And secondly good performance requires a few clusters with a lot of fanals in each, what is not biologically plausible.

In order to counterbalance those drawbacks, two solutions are described in this document. The first one consists in recursively dividing each cluster in subclusters with an associated added decoding rule. The second one considers sparse input messages.

5.3.1 Fractal approach

5.3.1.1 Pseudo-fractal idea

The proposed network is not suitable for a fractal approach. As a matter of fact, it associates neurons through cliques. Therefore a direct association of sparse networks through a high-level sparse network would need a definition of meta-cliques - that is: cliques of cliques. These considerations do not correspond to the current neurobiology literature.

A pseudo-fractal approach may nevertheless be applied. In order to consider smaller clusters without sacrificing performance, one can use a bottom-up pseudo-fractal approach as follows.[5]

Let us consider for instance a network with $c = 4$ clusters of $l = 256$ neurons each. This means that input messages, if binary, are composed of four sub-messages of length $\kappa = 9$. One can divide recursively each sub-message m_i in smaller pieces: $m = m_i^1 m_i^2$.

For more simplicity, let us suppose that m_i^1 is of length 2 and fanals in cluster i form a plane grid. If so, the fanals can be organized so that the value of bits in n_i are coding the physical emplacement of the message in the network, such as described in Figure 5.15.

This process can be iterated several times, embodying the fractal approach.

Splitting clusters can be accompanied by an added decoding rule: in each cluster one single sub-cluster must only be activated. This rule, which is the meta-version of the local decoding rule, echoes to neurobiology literature where one can read that the activity of a neocortex column is likely to prevent the activity of its neighbour ones. In other words, not only a single fanal is activated in a cluster at a time but also a unique sub-cluster can be activated in a cluster at a given time.

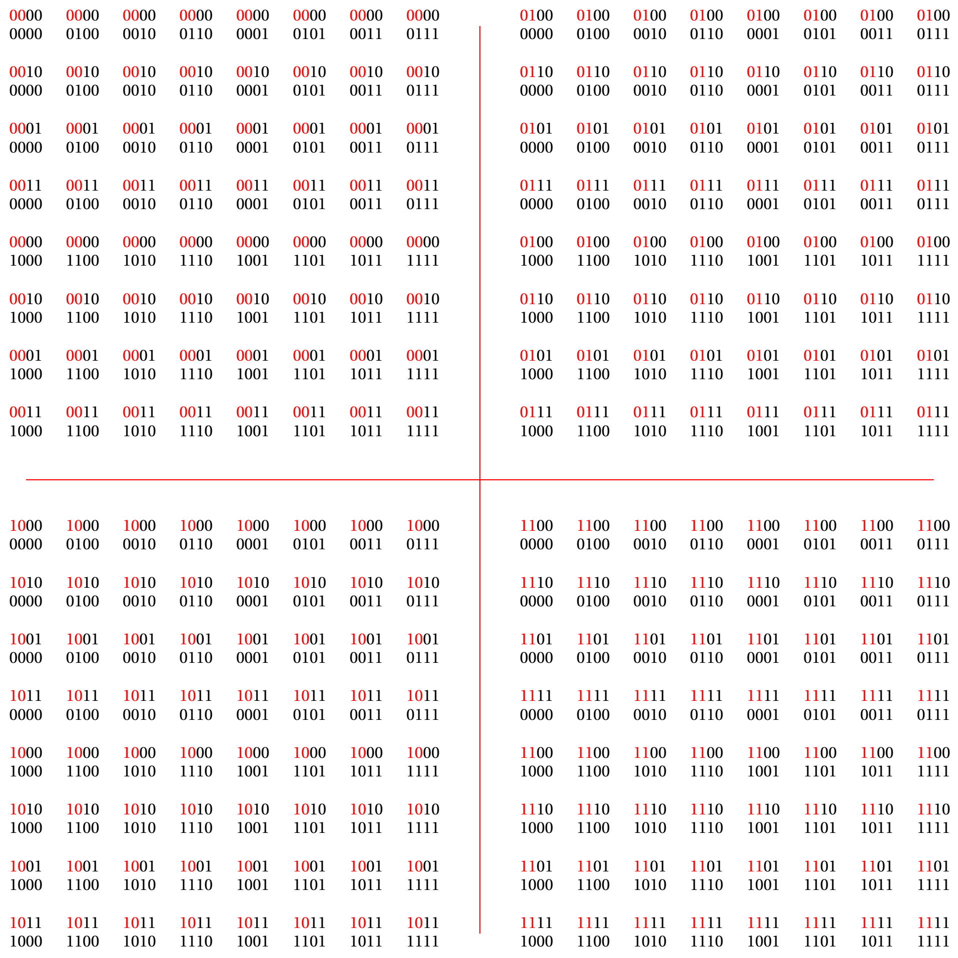
This construction does not affect the input message length, but reduces significantly the size of clusters with unmodified performance.

5.3.1.2 Partial erasure and performance

Actually, this approach also enables the consideration for partial erasures on clusters. It is now possible to erase part of a sub-message - that is for instance m_i^1 in m_i - with a very limited impact on performance.

In order to adapt the network to this partial erasure rule, a normalization rule has to be applied. As a matter of fact, erasing the part m_i^1 of a sub-message m_i

[5] We call this approach pseudo-fractal as it keeps the global decoding principle as it is.



0000 0000	0000 0100	0000 0010	0000 0110	0000 0001	0000 0101	0000 0011	0000 0111	0100 0000	0100 0100	0100 0010	0100 0110	0100 0001	0100 0101	0100 0011	0100 0111
0010 0000	0010 0100	0010 0010	0010 0110	0010 0001	0010 0101	0010 0011	0010 0111	0110 0000	0110 0100	0110 0010	0110 0110	0110 0001	0110 0101	0110 0011	0110 0111
0001 0000	0001 0100	0001 0010	0001 0110	0001 0001	0001 0101	0001 0011	0001 0111	0101 0000	0101 0100	0101 0010	0101 0110	0101 0001	0101 0101	0101 0011	0101 0111
0011 0000	0011 0100	0011 0010	0011 0110	0011 0001	0011 0101	0011 0011	0011 0111	0111 0000	0111 0100	0111 0010	0111 0110	0111 0001	0111 0101	0111 0011	0111 0111
0000 1000	0000 1100	0000 1010	0000 1110	0000 1001	0000 1101	0000 1011	0000 1111	0100 1000	0100 1100	0100 1010	0100 1110	0100 1001	0100 1101	0100 1011	0100 1111
0010 1000	0010 1100	0010 1010	0010 1110	0010 1001	0010 1101	0010 1011	0010 1111	0110 1000	0110 1100	0110 1010	0110 1110	0110 1001	0110 1101	0110 1011	0110 1111
0001 1000	0001 1100	0001 1010	0001 1110	0001 1001	0001 1101	0001 1011	0001 1111	0101 1000	0101 1100	0101 1010	0101 1110	0101 1001	0101 1101	0101 1011	0101 1111
0011 1000	0011 1100	0011 1010	0011 1110	0011 1001	0011 1101	0011 1011	0011 1111	0111 1000	0111 1100	0111 1010	0111 1110	0111 1001	0111 1101	0111 1011	0111 1111
1000 0000	1000 0100	1000 0010	1000 0110	1000 0001	1000 0101	1000 0011	1000 0111	1100 0000	1100 0100	1100 0010	1100 0110	1100 0001	1100 0101	1100 0011	1100 0111
1010 0000	1010 0100	1010 0010	1010 0110	1010 0001	1010 0101	1010 0011	1010 0111	1110 0000	1110 0100	1110 0010	1110 0110	1110 0001	1110 0101	1110 0011	1110 0111
1001 0000	1001 0100	1001 0010	1001 0110	1001 0001	1001 0101	1001 0011	1001 0111	1101 0000	1101 0100	1101 0010	1101 0110	1101 0001	1101 0101	1101 0011	1101 0111
1011 0000	1011 0100	1011 0010	1011 0110	1011 0001	1011 0101	1011 0011	1011 0111	1111 0000	1111 0100	1111 0010	1111 0110	1111 0001	1111 0101	1111 0011	1111 0111
1000 1000	1000 1100	1000 1010	1000 1110	1000 1001	1000 1101	1000 1011	1000 1111	1100 1000	1100 1100	1100 1010	1100 1110	1100 1001	1100 1101	1100 1011	1100 1111
1010 1000	1010 1100	1010 1010	1010 1110	1010 1001	1010 1101	1010 1011	1010 1111	1110 1000	1110 1100	1110 1010	1110 1110	1110 1001	1110 1101	1110 1011	1110 1111
1001 1000	1001 1100	1001 1010	1001 1110	1001 1001	1001 1101	1001 1011	1001 1111	1101 1000	1101 1100	1101 1010	1101 1110	1101 1001	1101 1101	1101 1011	1101 1111
1011 1000	1011 1100	1011 1010	1011 1110	1011 1001	1011 1101	1011 1011	1011 1111	1111 1000	1111 1100	1111 1010	1111 1110	1111 1001	1111 1101	1111 1011	1111 1111

Figure 5.15: Example of a disposition of fanals such that the two first bits of the sub-message they correspond to are coding their physical placement: 00 for top left, 01 for top right, 10 for bottom left and 11 for bottom right.

gives the possibility of an ambiguity on a large number of fanals in the corresponding cluster. Due to the decoding process, these fanals have the same activity as those from the clusters with perfect information. This means that the partially erased clusters will have a much more important impact - as they will have more activated fanals - than the perfect information ones on the final decision, what is clearly counterproductive.

So let us add to the network a normalization rule, which could correspond biologically to a distribution of energy among clusters. This rule states that the sum of all activities of fanals in a cluster cannot surpass 1. Figure 5.16 depicts the performance of this modified network when used as an associative memory and after four iterations[6]. It is compared with the initial network for a comparable amount of erased bits.

Actually, the worst case is when all erased bits are in the same cluster, leading to the same performance as the initial model (see Equation (5.12)).

One can see that in any case of partial erasures among several clusters the performance is slightly increased. We can actually explain this gain as when the erasures are shared among different clusters, the performance is benefiting from several iterations.

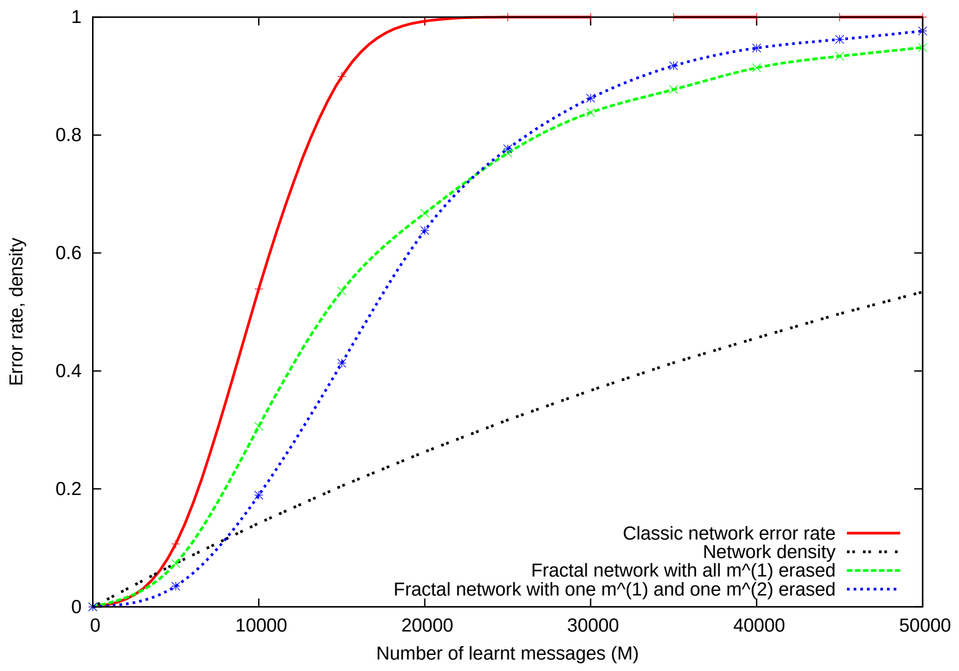


Figure 5.16: Comparison of performance of the classic network with the fractal one when erasing an equivalent amount of information (8 bits). These networks are composed of $c = 4$ clusters of $l = 256$ neurons each. The decoding process uses 4 iterations and the memory effect has been set to a large value.

5.3.2 Sparse networks of sparse messages

5.3.2.1 Construction

Though it presents an interest in virtually reducing the size of clusters, the described above method does not dissociate the length of learned messages from

[6]These results were obtained by simulations made by Xiaoran Jiang, Ph.D. student at Télécom Bretagne.

the size of the network. Moreover, the network has always an active fanal in every meta-cluster, contradicting the observations made by neurologists that specific regions in the brain are associated with different stimulations.

In order to dissociate the length of messages and the number of neurons, one can consider learning sparse messages (that is shorter messages).

However, the proposed network is not directly adapted for the learning of sparse messages. As a matter of fact, the network requires every cluster to select a fanal and therefore forces all characters to be retrieved.

Yet it is possible to slightly modify the dynamic rules of the network so that it becomes adapted to sparse messages. Actually, the learning process is not modified. Thus, the clique printed in the network after the learning of a message is that containing all the fanals corresponding to its characters.

In order to enable the retrieving of messages, we have to restrict the set of possible messages. Note that with each message is associated a unique sub-network where only the clusters corresponding to non-absent characters are considered. By forcing sub-networks to satisfy some constraints it is possible to retrieve sparse messages with no need for modifying deeply the network principles.

The constraint we propose is that distinct sub-networks should not share more than a single cluster in common. Also, for more simplicity, we force the different sub-networks to address the same number of clusters. In other words, if we index each cluster by a number and represent a sub-network by a binary word that contain "1" at the locations corresponding to the cluster it addresses - "0" elsewhere -, the sub-networks will form a constant-weight code with overlapping $r = 1$.

There are two reasons why this constraint is a good choice - though it is not likely to be the best one. The first one is that any sub-network can be unambiguously retrieved given only two clusters. The second one is that the density between two given clusters is only affected by the learnings on a single sub-network.

Figure 5.17 depicts an example of a list of sub-networks compatible with the constraint previously defined.

5.3.2.2 Decoding process

During the decoding process, the network must identify without ambiguity the sub-network a partially erased sparse message is addressed to. As sub-networks may share up to a single cluster in common, the partial information has to contain at least information on two clusters. This is not really a restriction as trying

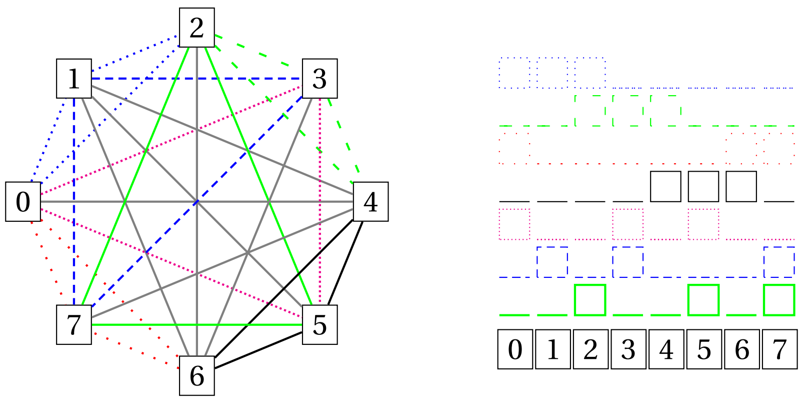


Figure 5.17: Example of sub-networks made of 3 clusters each such that they do not share more than one cluster in common.

to retrieve messages having only pieces of information in a single cluster would limit dramatically the number of learned messages (to at most l).

To recognize whether a cluster should be activated or not - whether it is part of the sub-network or not -, we add a time diversity to the network. Actually, we use the fact that if clusters are not characteristic to sub-networks, connections on the other hand specify two clusters each and are therefore characteristic to sub-networks. In other terms, one can add to connections a temporal signature associated with the sub-network it corresponds to (that is: use an edge-coloured graph).

For instance, let us consider that every sub-network s_i is characterized by a time propagation t_i. If the network is initialized with pieces of information on two clusters belonging to sub-network s_i, other clusters in s_i will receive two incoming signals - one for each cluster provided with information - at the same time (that is t_i). On the other hand, some other clusters will possibly receive two signals but, if so, at different times. Indeed, if they receive signals at the same time, this means that those two clusters belong to the same sub-network, and as pairs of clusters are characteristic of sub-networks, it necessarily corresponds to s_i.

Thus, adding a minimal activity condition on fanals (corresponding to threshold $\sigma = 2$) and forcing time coincidence in order to aggregate different inputs, the modified network is able to select and restrain its activity to the correct sub-network. In addition, note that the amount of memory necessary to store those propagation times is negligible and will therefore not have a significant impact on efficiency. These considerations for synchronisation echo to neurobiological literature [8].

5.3.2.3 Performance

As already explained, sub-networks form a constant weight code with length c, weight c' and overlapping $r = 1$. According to section 4.2.2, one may find up to $\frac{c(c-1)}{c'(c'-1)}$ such sub-networks if c' is a prime number and c a power of c'.

Using Equation 5.12, if l is large enough and as a first approximation, the number of messages our network can learn is

$$M = \alpha l^2$$

where $\alpha > 0$. If the network is made of c clusters, this equation may be rewritten as follows:

$$M = \alpha \left(\frac{n}{c}\right)^2$$

With the same total of neurons, and if one divide the size of clusters by c' (that is multiply the number of clusters by c'), this number is reduced to:

$$M_{c'} = \alpha \left(\frac{n}{cc'}\right)^2$$

But on the other hand, rather than learning messages in the whole network, one can prefer learning messages on the sub-networks previously defined of c clusters each. They will then all be able to learn $M_{c'}$ messages each, leading to a total of:

$$\begin{aligned} M_{sparse} &= \frac{cc'(cc'-1)}{c(c-1)}\alpha \left(\frac{n}{cc'}\right)^2 && (5.16) \\ &\approx \alpha \left(\frac{n}{c}\right)^2 && (5.17) \end{aligned}$$

It is thus possible to maintain the same number of learned messages while decreasing l (and increasing c accordingly) if the latter are sparse.

Actually, by giving a closer look to Equation (5.13), it can be observed that l has a double impact on performance: it decreases the probability of error as it grows since the density decreases, but it also increases the former since there are more competition between the fanals in the clusters. In other words, when one cluster in the network is not provided with information, it corresponds to a bigger erasure on the input if l is larger.

Considering this property, one can expect to increase diversity when considering sparse messages. Simulations support this remark: with $c = 5$ *clusters of* $l = 2500$ *neurons, and considering a division of clusters in* $c' = 25$ *parts each, the diversity is multiplied by a factor of* 3.2 *and capacity by a factor of* 2.5 *with the same error probability.*

6 Conclusion, opening

6.1 Conclusion

We have considered different ways to embed coding techniques into neural networks in order to increase their performance as associative memories. First we have shown that cliques are easily controllable entities that are perfect candidates for the storage of pieces of information in neural networks. We also proposed a very simple way to associate pieces of information with virtual cliques giving a much better performance than the state-of-the-art HNN.

Then we showed that several codes can be efficiently decoded using neural networks: constant weight codes and c-cliques. We presented a way to introduce codes in the heart of the HNN dynamic, which considerably increased the number of learnt messages as well as the retrieving performance.

Finally, we introduced an original network that almost achieves optimality. It uses c-cliques and constant weight codes as the heart of its learning and retrieving scheme. It can be easily adapted to be scalable and to present efficient performance on correlated messages.

Associative memories give an answer to just a very small part of the question of the storage of pieces of information in the brain. They do not consider computation, and they would not be in any case sufficient to model a brain mimetic device. Nevertheless, they seem to be at the very heart of the human capability to store, retrieve and, above all, combine pieces of information. As far as those aspects are concerned, the model described in this document is extremely satisfying in various aspects.

An aspect of prime importance is the biological plausibility. From this point of view, the improvements given by the network presented in Chapter 5, by comparison with the HNN, are sizeable.

Neurons are binary, like in the HNN, but with positive values (0 or 1). They are grouped into clusters having fanals with similar activity (*i.e.* addressing the same sub-message). Those clusters are linked one to each other through neural cliques, producing a sparse coding of the impulse. The number of messages learnt can be made independent of the number of neurons (with the limit of the efficiency), as well as from their size. Actually, one should not consider drawing an analogy directly between fanals and neurons in the neocortex. A more plausible comparison would be to associate fanals with mini-columns, each of them being seen as a specific machinery implementing the various roles of fanals.

The dynamic of the network is completely compatible with some observations recently published in the neurobiology literature. For instance, in [42] is described an experience in which the neurons of a cat are excited. This leads to the activity of a set of other neurons that completely change when a neighbour neuron is excited instead. The same experience on our network would lead to the same observation: the activation of a fanal would activate a clique completely different from the one we would obtain by activating another fanal.

The network is sparse, and connections are also binary (0 or 1). Those connections are resistant to noise, and coherent with the Hebbian rule. The activity can be restricted to several clusters, possibly in a neighbourhood, offering a large number of openings on the way information is carried over the different clusters. Finally, the dynamic of the retrieving process requires a very few iterations, either in synchronous or asynchronous ways, and provide the network with an increasingly precise idea of its convergence state.

Yet the bidirectional connections are still a major drawback. They do not correspond to any plausibility. The possibility to consider unidirectional connections instead has already been considered, and should appear in an upcoming thesis. Unidirectionality allows the network to learn sequence of sub-messages, providing the network with a temporal dimension. They also allow it to learn arbitrary long messages (even longer than the number of neurons), using the same clusters at different times during the learning and decoding process. Figure 6.1 depicts this idea.

Another aspect of prime importance is the performance of the network.

Concerning the diversity, the number of messages the network can learn grows quadratically with its size, leading to the upper-bound introduced in the first section. Compared to the state of the art, this is a dramatic increase as the gain is asymptotically infinitely large. On typical sizes (several hundreds of neurons), it corresponds to a factor of several hundreds.

Regarding capacity, the proposed networks are able to surpass efficiency 1, leading to a compression of the inputs. This ability is far better than that of the

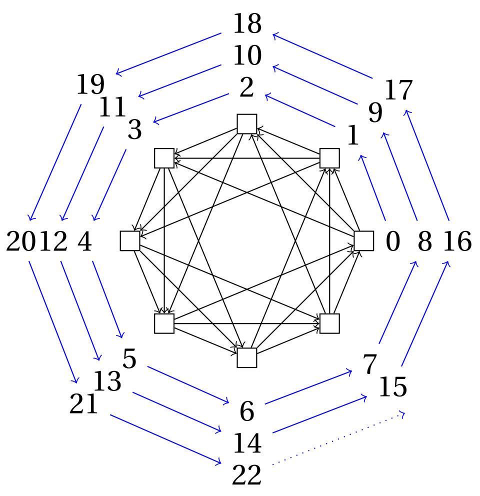


Figure 6.1: Use of unidirectional connections to learn arbitrary long sequential messages

HNN which tends to zero as the number of neurons tends to infinity. It allows the proposed network to be a very interesting candidate for new memories based on associative functioning.

> *The main interest of the work described in this document is that it shows one does not need to find a trade-off between biological plausibility and efficiency. Both aspects can be plainly satisfied without any conflict.*

6.2 Openings

A very interesting opening consists of pointing out that the network presented in Chapter 5 is ready to learn any kind of data. This original memory device arises a lot of openings on how these intertwined representations of information could enable cross breeding or production of new one.

There is still a lot of work to be done, all the more since this memory is at the junction of three main domains: neurobiology, cognitive science and computer science. Some openings are described in Figure 6.2.

They are classified into three main concepts.

6.2.1 Biological aspects

To bring our model closer to biological aspects, the neurons should be considered with the spiking neuron model. This model is considered by a large part

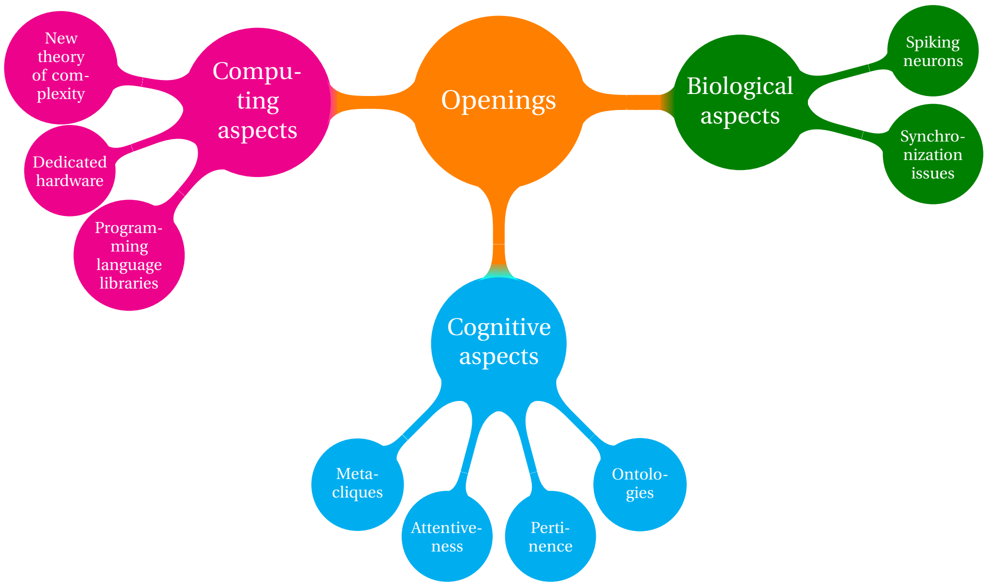


Figure 6.2: Mind map of openings.

of the community to be rich enough to allow parallels between neurobiological observations and artificial networks dynamics.

The last improvement presented in this document adding sparsity to the messages also corresponds to some recent papers where synchronisation is of importance and should be investigated further [8].

Up to now, we regarded signals between neurons as binary and non impulsional. It is quite different in the cortex where neurons share information in the form of spikes. There are two ways to respond to this observation, depending on whether one considers the arrival times of these pulses as critical or not (actually these two standpoints correspond to two distinct schools of thought in neurobiology). If we assume that synapses keep the effects of the incident spikes in brief memory and that the spikes arrive in a time window long enough to enable the summation of these effects, then the problem can be considered as a static one. On the other hand, if the times of arrival of multiple spikes are important in the decision-making of neurons, we have to add a temporal dimension to our neural model. This could be a very interesting opening since introducing a space-time filtering constraint on the neuronal activity could lead to a powerful way to select a particular clique among a huge number.

We have also to remember that the brain is not a deterministic machine. Neuronal noise is assuredly one of the reasons that explain this. Noise can be very easily introduced in the proposed networks, in various ways, either on the synaptic weights (ideally 0 or 1) or on the activity thresholds of neurons and clusters. In the space-time version of the model, noise could also affect propagation delays. It is obvious that in such situations, noise can play an important role that could explain why the recall of a particular learnt piece of information from a distorted version of it can be very erratic.

The network model we advocate lends itself to the study of neural wave propagation as brought to the fore by some recent experiments [42]. This undulatory behaviour is that of a non linear dynamic system with a huge number of variables. It is then impossible to analyse it with mathematical tools; in return, it could be simulated as a propagation medium (cliques can propagate their activity through overlapping vertices and edges) and interpreted: it will be very interesting to discover what the physical characteristics of this medium are, from a macroscopic point of view.

6.2.2 Computing aspects

The effectiveness of these associative memories make them perfect candidates for an alternative to index-based classical memories. In the field of computer science, a lot of algorithms would benefit from a dedicated hardware able to realise such operations in a restricted time.

This would imply a lot of open work, from the design of circuits to the formalisation of the notion of associative memories, and the consideration of library to interface programming languages with dedicated hardware. A library has already been programmed during the thesis and its documentation is available in Appendix C.

Other improving aspects would include the consideration of distorted messages. If we want the networks to be able to recover such messages or situations, we need to relax the winner-take-all rule and propose a new one giving the possibility of activating a small population within each cluster, according to some vicinity law yet to be discovered. This is a typical problem in error correcting decoders related to the concept of soft input. A soft input does not impose a value (either 0 or 1) but gives a probability on this value. In the neural network, a soft input would excite one particular neuron but also excite its neighbours, to a lesser degree.

6.2.3 Cognitive aspects

If one wants to dig further into the possibility to develop a machine able to manipulate information and based on these memories, he would have to consider a lot of open questions.

First of all, it is clear that this machines lack objectives, which should be related to the notions of attentiveness, or pertinence. Moreover, the ability to learn higher-order data would require to consider the learning of meta-cliques, which has not been investigated so far.

For the need of comparison with the state of the art, we have so far considered binary messages. Since we want to design cognitive machines, the binary messages have to be replaced with cognitive messages. This opens questions on the definition of those cognitive messages (writing, meaning, strength, class, etc.) that these networks will be able to store and process. Such questions will have to be addressed in the light of the knowledge sciences outcomes, in particular ontology metadata.

Another, and probably the most important point is about the data structure properties. Unlike classical memories in which messages are stored in separate physical places, these networks store overlapping messages (the cliques share vertices and/or edges). Therefore, the recall of a particular message has a certain influence on closely related cliques, that is, other pieces of information which can become relevant under conditions to investigate.

Appendices

A
Efficiency gain with non-ordering of messages

Let us show that the non-ordering of messages can result in an arbitrarily large efficiency, even when no message has been learnt a large number of times.

First, let us recall that the efficiency is measured as the ratio between the amount of information learnt to the amount of information used to store the messages.

We introduce the prefix tree acceptor [43] associated with a set of messages. This tree, which is a particular automaton, is the smallest one (in terms of number of nodes) that recognize exactly these messages.

For instance, the prefix tree acceptor associated with the learning set

$$\{00000, 01000, 10000, 01001, 01110\}$$

made of $M = 5$ messages of length $n = 5$ is depicted in Figure A.1.

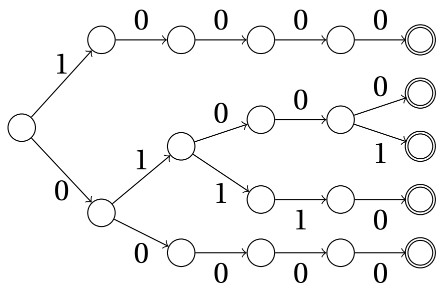


Figure A.1: Example of the prefix tree acceptor associated with the learning set $\{00000, 01000, 10000, 01001, 01110\}$. Accepting states are denoted by double lines and transitions are labelled by symbols (either 0 or 1).

To encode this structure into a classical memory, we simply list all branches using the following coding system: only the values from the first connection that differ from the previously listed branch are denoted (all are denoted if it is the first branch). So, for the prefix tree depicted in Figure A.1, the result of the encoding is:

```
10000
01000
1
110
0000
```

For instance, the two lines

```
01000
1
```

mean that the tree contains the branch `01000` and the branch that is the same with the difference that the last symbol is `1`, that is: `01001`.

Using this coding system, the number of needed symbols is simply the sum of C the number of connections in the prefix tree and as many newline symbols as messages, that is a total of $C + M$ symbols. These symbols, in the case of binary messages, are encoded over 3 values: $\{0; 1\}$ and the newline symbol. Thus this encoding requires a total of $(C + M)\log_2(3)$ bits. This value has to be compared with the amount of information learnt, that is: Mn bits.

If messages are i.i.d., the number of expected connections can be estimated as follows: the probability that a given connection has been used is the probability that a message matches the associated prefix. If this connection is at depth k, then this probability is:

$$P_k(M) = 1 - (1 - (2^{-k}))^M$$

Since there are 2^k distinct possible connections at depth k, the expected number of connections is:

$$C(n, M) = \sum_{k=1}^{n} 2^k P_k(M)$$

Figure A.1 shows the evolution of the ratio $\frac{Mn}{C(n,M)}$, that is the expected efficiency of the prefix tree acceptor, in function of the number of learnt messages M, and for various sizes of messages n. Note that the probability that a message is present two times in the learning set is less than 0.1 in all cases. We can see that the efficiency can be made larger than 1 using prefix tree acceptors.

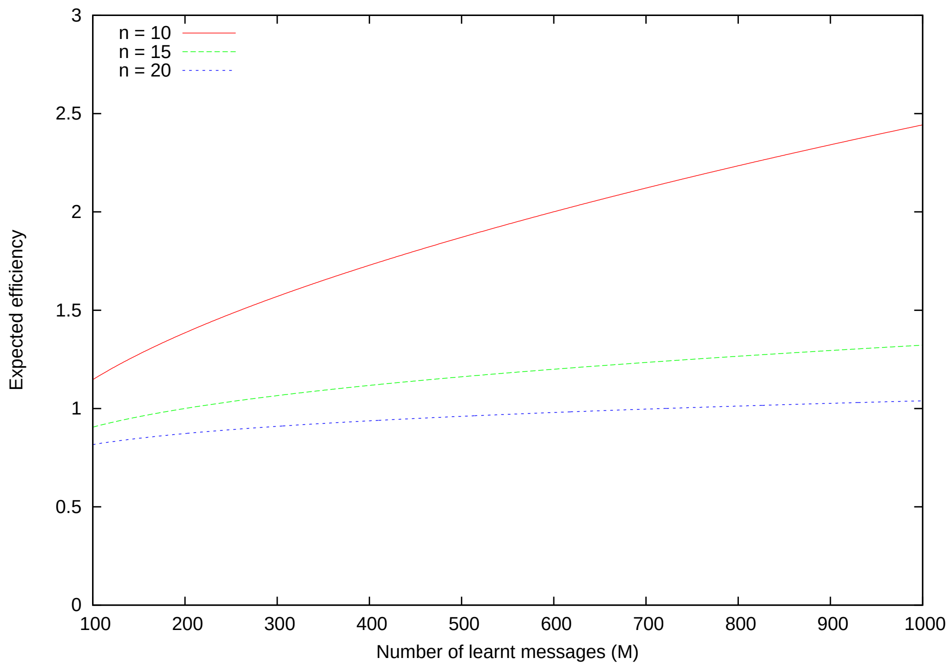


Figure A.2: Expected efficiency of prefix tree acceptors after the learning of M i.i.d. binary random messages.

B
Optimal construction of constant weight codes

The next paragraphs present an upper bound for the number of elements in a constant weight code of length n, weight w and overlapping r. Then a constructive lower-bound that merges the upper one in some cases is presented.

The estimation of an upper bound is trivial: let us consider a code $C_{cw}(n,w,r)$. The constraint on the distance can be rewritten as follows: a codeword $c \in C_{cw}(n,w,r)$ defines $\binom{w}{r+1}$ constraints corresponding to as many subsets of $r+1$ symbols that cannot be found in another codeword. In other terms, if a word contains $r+1$ characters at specific locations, no other word can contain the same ones at the same locations and each word defines $\binom{w}{r+1}$ such constraints.

On the other hand, the total number of available constraints is $\binom{n}{r+1}$.

Therefore an upper-bound for the number of codewords in a code $C_{cw}(n,w,r)$ is:

$$\boxed{U^b_{C^(_{cw}n,w,r)} = \prod_{i=0}^{r} \frac{n-i}{w-i}} \qquad \text{(B.0)}$$

This upper bound is fairly accurate as we shall develop in the following paragraphs. It is interesting to point out that once again a good compromise has to be found between minimum distance and rate of the code.

The lower bound is much more complicated to estimate. The following paragraphs give a constructive top achieving lower bound in some specific cases.

Let us consider as a first constraint that w is a prime number, that the code is binary, that $r = 1$ and then let us add that $\exists p, n = w^p$. Under those conditions, the lower bound described beneath is achieving the upper bound.

To construct our words, we will consider a recursive method that will consist in p steps. At each step, words are being added to a growing set of already considered codewords such that they remain all together consistent with the constraints of the constant weight code.

At step 1, the codewords that are being added to the code are those in:

$$S_1 = \{0^{iw}1^w0^{n-w(i+1)}, 0 \leq i < \frac{n}{w}\} \tag{B.1}$$

Those codewords have no overlapping and therefore are trivially verifying the constant weight constraint.

At step $q+1$, words contain their non-zero characters on predetermined packets of w^{q+1} bits. More precisely, the words added in S_{q+1} are also in $\{0^{iw^{q+1}}(0+1)^{w^{q+1}}0^{n-(i+1)w^{q+1}}, 0 \leq i < \frac{n}{w^{q+1}}\}$. Those are not in conflict with the words previously defined:

$$S_{q+1} = \{0^{iw^{q+1}}S'_{q+1}0^{n-(i+1)w^{q+1}}, 0 \leq i < \frac{n}{w^{q+1}}\} \tag{B.2}$$

Where:

$$\begin{aligned} S'_{q+1} &= \Bigg\{ \prod_{v=0}^{w^{q+1}} 0^{(v*j+l \pmod{w^{q+1}})}10^{w^{q+1}-1-(v*j+l \pmod{w^{q+1}})}, \\ &\quad 0 \leq j < w^{q+1}, 0 \leq l < w^{q+1} \Bigg\} \end{aligned} \tag{B.3}$$

It would be fastidious to prove the correctness of this construction. At the end, the code is obtained as the disjoint union of the different S_i:

$$S = \bigcup_{q=1}^{p} S_q \tag{B.4}$$

Thus, the number of codewords is:

$$\boxed{\begin{aligned} \#S &= \sum_{q=1}^{p} \frac{n}{w^q} w^{2q-1} \\ &= \frac{n}{w}\frac{1-w^p}{1-w} \\ &= \frac{n(n-1)}{w(w-1)} \end{aligned}} \tag{B.2}$$

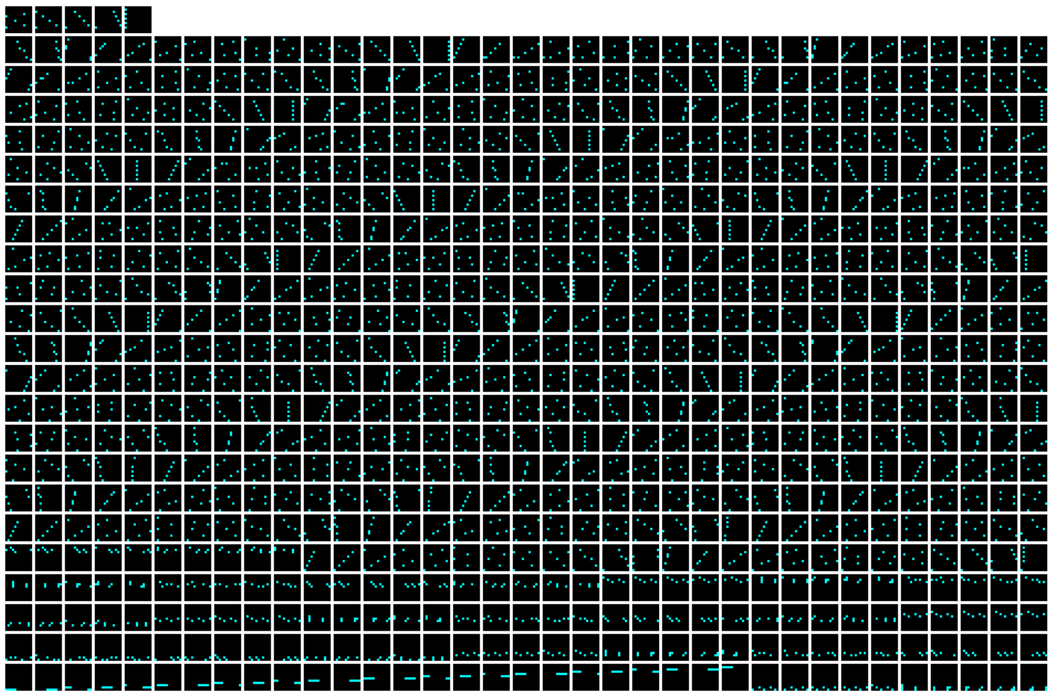


Figure B.1: Representation of the 775 codewords obtained with the previously described construction and being part of a constant weight code with $n = 125$, $w = 5$ and $r = 1$. Codewords are represented by large black squares, "1" in a codeword by a lighter small square. This construction is optimal.

Thus we are able to build, for $r = 1$, w a prime number and $\exists p, n = w^p$, a constant weight code that achieves optimality as far as the rate of the code is concerned (or indifferently the merit factor). This result is used in Chapter 5.

Figure B.1 depicts the 775 codewords obtained with this method when $n = 125$, $w = 5$ and $r = 1$. This construction is optimal.

C OCaml library for our proposed associative memories

This appendix contains the documentation of a library that was developped during the thesis. This library is available for OCaml[1] and is licensed under GPL. It can be downloaded from the Caml-Hump[2].

The library is provided with an example of use corresponding to the curve depicted in Figure 5.9.

C.1 Module `Sam` : Sparse Associative Memories

The module Sam allows common operations on sparse associative memories according to the model introduced in Chapter 5. They store tuples of elements (called messages of characters) such that they are able to retrieve them given only partial information with a good probability. This implementation allows to consider associative memories of any type. On the other hand, performance is limited: it can be 10 times longer than using an ad hoc construction (it uses hash tables at multiple locations). A typical use is:

- `# let sam = Sam.create 4;;`

  `val sam : '_a Sam.t = Sam.Sam [|<abstr>; <abstr>; <abstr>; <abstr>|]`

- `# Sam.add sam [|"Hel";"lo";" Wo";"rld!"|];;`

  `- : unit = ()`

- `# Sam.retrieve_unique sam [|None; Some "lo"; None; Some "rld!"|];;`

  `- : string array = [|"Hel"; "lo"; " Wo"; "rld!"|]`

[1]OCaml is a programming language developped at INRIA.
[2]http://caml.inria.fr/cgi-bin/hump.cgi?contrib=762

- `# Sam.add sam [|"H";"i";" Wo";"rld!"|];;`

  ```
  - :  unit = ()
  ```

- `# Sam.retrieve sam [|None; None; Some " Wo"; Some "rld!"|];;`

  ```
  - :  string list array = [|["H"; "Hel"]; ["i"; "lo"]; [" Wo"]; ["rld!"]|]
  ```

```
type 'a t = {
  mutable max_iterations : int ;
  mutable total_iterations : int ;
  mutable nb_retrievals : int ;
  size : int ;
  length : int ;
  table : ('a, ('a, bool) Hashtbl.t array) Hashtbl.t array ;
  lists : ('a, 'a list array) Hashtbl.t array ;
}
```

Type for internal representation of sparse associative memories.

```
val create : ?length:int -> int -> 'a t
```

`Sam.create n` creates an associative memory working on n characters.
`Sam.create ~length:l n` creates an associative memory working on n characters with the information that the number of different characters will be about l.

```
val size : 'a t -> int
```

`Sam.size sam` returns the number of characters in messages in sam.

```
val add : 'a t -> 'a array -> unit
```

`Sam.add sam motive` adds message `motive` to the sam described by `sam`.
Learning a second time the same message will leave the sam unchanged.

```
val retrieve_from_lists :
  'a t -> ?iterations:int -> 'a list array -> 'a list array
```

`Sam.retrieve_from_lists ?iterations sam listed_motives` takes an array of lists of characters as argument, which contains the possibilities at each position. The list `[]` is interpreted as being all possibilities for associated character. It iterates until reaching a fixed point unless a maximum number of iterations is given.

```
val retrieve : 'a t -> ?iterations:int -> 'a option array -> 'a list array
```

`Sam.retrieve sam some_motive` considers `None` values as characters to be retrieved. It is equivalent to `Sam.retrieve_from_lists sam listed_motives` when `listed_motives` is the same array as `some_motive` where `Some x` has been replaced by `[x]` and `None` by `[]`.

```
exception Not_unique
```

Exception raised when finding several solutions in the decoding of `retrieve_unique`.

```
val retrieve_unique : 'a t -> ?iterations:int -> 'a option array -> 'a array
```

`Sam.retrieve_unique sam some_motive` is the same as `Sam.retrieve` but it gives as a result the unique retrieved pattern. If several are decoded, it raises exception `Not_unique`.

```
val alphabet : 'a t -> int -> 'a list
```

`Sam.alphabet sam i` returns the list of all used characters at position `i` in `sam`.

```
val density :
  ?length:int -> 'a t -> float
```

`Sam.density sam` returns the density of `sam` considering only used characters. `Sam.density ~length=l sam` returns the density of `sam` considering that characters can take `l` different values.

```
val average_iterations : 'a t -> float
```

`Sam.average_iterations sam` returns the average number of iterations during the previously retrievals realized on sam.

```
val max_iterations : 'a t -> int
```

`Sam.max_iterations sam` returns the maximum number of iteration observed for a single retrieval previously realized on sam.

List of Figures

List of Tables

Bibliography

[1] T. G. Dietterich and G. Bakiri, “Solving multiclass learning problems via error-correcting output codes,” *J. Artif. Int. Res.*, vol. 2, no. 1, pp. 263–286, 1994.

[2] J. Bruck and M. Blaum, “Neural networks, error-correcting codes, and polynomials over the binary n-cube,” *IEEE Trans. on Information Theory*, vol. 35, no. 5, pp. 976–987, 1989.

[3] L. Euler, “E53 – solutio problematis ad geometriam situs pertinentis,” *Commentarii academiae scientiarum Petropolitanae 8*, pp. 128–140, 1741.

[4] V. Gripon and O. Serre, “Qualitative concurrent stochastic games with imperfect information,” *Automata, Languages and Programming, 36th Internatilonal Collogquium*, vol. 5556, pp. 200–211, Rhodes, Greece, Jul. 2009.

[5] J. B. Furness, “Types of neurons in the enteric nervous system,” *Journal of the Autonomic Nervous System*, vol. 81, no. 1-3, pp. 87–96, Jul. 2000.

[6] M. E. J. Newman, “The structure and function of complex networks,” *SIAM Review*, vol. 45, pp. 167–256, 2003.

[7] L. Lin, R. Osan, and J. Z. Tsien, “Organizing principles of real-time memory encoding: neural clique assemblies and universal neural codes,” *Trends Neurosci*, vol. 29, no. 1, pp. 48–57, 2006.

[8] D. F. M. Goodman and R. Brette, “Spike-timing-based computation in sound localization,” *PLoS Comput Biol*, vol. 6, no. 11, p. e1000993, 11 2010.

[9] G. J. Rinkus, “A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality,” *Frontiers in Neuroanatomy*, vol. 4, no. 17, Jun. 2010.

[10] D. K. Lee, L. Itti, C. Koch, and J. Braun, “Attention activates winner-take-all competition among visual filters,” *Nature Neuroscience*, vol. 2, no. 4, pp. 375–81, Apr. 1999.

[11] R. L. Coultrip, R. H. Granger, and G. Lynch, “A cortical model of winner-take-all competition via lateral inhibition,” *Neural Networks*, vol. 5, pp. 47–54, 1992.

[12] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, “Winner-take-all networks of o(n) complexity,” in *Advances in Neural Information Processing Systems.* Morgan Kaufmann, 1989, pp. 703–711.

[13] C. Möller, J. Lücke, J. Zhu, P. M. Faustmann, and C. von der Malsburg, "Glial cells for information routing?" *Cognitive Systems Research*, vol. 8, pp. 28 – 35, 2007.

[14] J. E. Niven and S. B. Laughlin, "Energy limitation as a selective pressure on the evolution of sensory systems," *Journal of Experimental Biology*, vol. 211, no. 11, pp. 1792–1804, Jun. 2008.

[15] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review*, vol. 63, pp. 81–97, 1956.

[16] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current opinion in neurobiology*, vol. 14, no. 4, pp. 481–487, Aug. 2004.

[17] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Neurocomputing: foundations of research*, pp. 509–521, 1988.

[18] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[19] Y. L. Cun, "A theoretical framework for back-propagation," 1988.

[20] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive science 9*, pp. 147–169, 1985.

[21] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational properties," *Proc. Natl Acad. Sci., Biophysics*, vol. 79, pp. 2554–2558, USA, 1982.

[22] R. Rojas, *Neural networks: a systematic introduction.* Springer-Verlag, 1996. [Online]. Available: http://books.google.com/books?id=txsjjYzFJS4C

[23] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Trans. Inf. Theor.*, vol. 33, no. 4, pp. 461–482, 1987.

[24] A. Storkey, "Increasing the capacity of a Hopfield network without sacrificing functionality," *ICANN'97*, pp. 451–456, Lausanne, Oct. 1997.

[25] Y. L. Cun, J. S. Denker, and S. A. Solla, "Advances in neural information processing systems 2," D. S. Touretzky, Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. Optimal brain damage, pp. 598–605.

[26] M. H. Histed, V. Bonin, and R. C. Reid, “Direct Activation of Sparse, Distributed Populations of Cortical Neurons by Electrical Microstimulation,” vol. 63, no. 4, pp. 508–522, Aug. 2009.

[27] A. P. Georgopoulos, R. E. Kettner, and A. B. Schwartzb, “Primate motor cortex and free arm movements to visual targets in three-dimensional space. ii. coding of the direction of movement by a neuronal population,” *Journal of Neuroscience*, vol. 8, pp. 2928–2937, 1988.

[28] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey, “On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex,” *J. Neurosci.*, vol. 2, no. 11, pp. 1527–1537, Nov. 1982.

[29] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” *Proc. of IEEE ICC ’93*, vol. 2, pp. 1064–1070, Geneva, May 1993.

[30] D. J. MacKay and R. M. Neal, “Good codes based on very sparse matrices,” *Cryptography and Coding. 5th IMA Conference, number 1025 in Lecture Notes in Computer Science*, pp. 100–111, Berlin, 1995.

[31] R. Gallager, “Low-density parity-check codes,” *IEEE Trans. on Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[32] F. J. MacWilliams and N. J. A. Sloane, “The theory of error-correcting codes,” pp. 526–527, North-Holland, 1979.

[33] J. F. Yang and C. M. Chen, “Winner-take-all neural networks using the highest threshold,” *IEEE Trans. Neural Netw.*, vol. 11, no. 1, Jan. 2000.

[34] A. J. Yu, M. A. Giese, and T. A. Poggio, “Biophysiologically plausible implementations of the maximum operation,” *Neural Comput.*, vol. 14, pp. 2857–2881, Dec. 2002.

[35] C. Berrou and V. Gripon, “Coded Hopfield networks,” *Proc. of 6" Int’l Symposium on Turbo Codes and Iterative Information Processing*, pp. 1–5, Brest, France, Sep. 2010.

[36] V. Gripon and C. Berrou, “Sparse neural networks with large learning diversity,” *IEEE transactions on neural networks*, vol. 22, no. 7, pp. 1087 – 1096, Jul. 2011.

[37] A. J. Yu, M. A. Giese, and T. A. Poggio, “Biophysiologically plausible implementations of the maximum operation,” *Neural Comput.*, vol. 14, pp. 2857–2881, Dec. 2002.

[38] J. Anderson, “Learning in sparsely connected and sparsely coded systems,” Working note, Aug. 2005.

[39] F. Schwenker, F. Schwenker, F. Schwenker, F. T. Sommer, F. T. Sommer, G. Palm, and G. Palm, “Iterative retrieval of sparsely coded associative memory patterns,” *Neural Networks*, vol. 9, pp. 445–455, 1995.

[40] S. ichi Amari, “Characteristics of sparsely encoded associative memory,” *Neural Networks*, vol. 2, no. 6, pp. 451 – 457, 1989.

[41] V. Gripon and C. Berrou, “A simple and efficient way to store many messages using neural cliques,” *Proc. of IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain*, pp. 54 – 58, Apr. 2011.

[42] M. H. Histed, V. Bonin, and R. C. Reid, “Direct Activation of Sparse, Distributed Populations of Cortical Neurons by Electrical Microstimulation,” no. 4, pp. 508–522, Aug. 2009.

[43] D. Angluin, “Inference of reversible languages,” *J. ACM*, vol. 29, pp. 741–765, Jul. 1982.