Coding Theory and Neural Associative Memories with Exponential Pattern Retrieval Capacity

THÈSE N^{O} 6089 (2014)

PRÉSENTÉE LE 17 AVRIL 2014

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE D'ALGORITHMIQUE
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Amir Hesam SALAVATI

acceptée sur proposition du jury:

Prof. W. Gerstner, président du jury
Prof. M. A. Shokrollahi, directeur de thèse
Prof. G. Kramer, rapporteur
Prof. H.-A. Loeliger, rapporteur
Prof. M. Seeger, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2014

Dedicated to my mother, father, brother and wife
for their unconditional and everlasting love, support and encouragements.

Abstract

The mid 20th century saw the publication of two pioneering works in the fields of neural networks and coding theory, respectively the work of McCulloch and Pitts in 1943, and the work of Shannon in 1948. The former paved the way for artificial neural networks while the latter introduced the concept of channel coding, which made reliable communication over noisy channels possible.

Though seemingly distant, these fields share certain similarities. One example is the neural associative memory, which is a particular class of neural networks capable of memorizing (learning) a set of patterns and recalling them later in the presence of noise, i.e., retrieving the correct memorized pattern from a given noisy version. As such, the neural associative memory problem is very similar to the one faced in communication systems where the goal is to reliably and efficiently retrieve a set of patterns (so called codewords) from noisy versions.

More interestingly, the techniques used to implement artificial neural associative memories look very similar to some of the decoding methods used in modern graph-based codes. This makes the pattern retrieval phase in neural associative memories very similar to iterative decoding techniques in modern coding theory.

However, despite the similarity of the tasks and techniques employed in both problems, there is a huge gap in terms of efficiency. Using binary codewords of length n, one can construct codes that are capable of reliably transmitting 2^{rn} codewords over a noisy channel, where $0 < r < 1$ is the code rate. In current neural associative memories, however, with a network of size n one can only memorize $O(n)$ binary patterns of length n. To be fair, these networks are able to memorize any set of *randomly* chosen patterns, while codes are carefully constructed. Nevertheless, this generality severely restricts the efficiency of the network.

In this thesis, we focus on bridging the performance gap between coding techniques and neural associative memories by exploiting the inherent structure of the input patterns in order to increase the pattern retrieval capacity from $O(n)$ to $O(a^n)$, where $a > 1$. Figure 1 illustrates the idea behind our approach; namely, it is much easier to memorize more patterns that have some redundancy like natural scenes in the left panel than to memorize the more random patterns in the right panel.

Figure 1: Which one is easier to memorize? Van Gogh's natural scenes or Picasso's cubism paintings?

More specifically, we focus on memorizing patterns that form a subspace (or more generally, a manifold). The proposed neural network is capable of learning and reliably recalling given patterns when they come from a subspace with dimension $k < n$ of the n-dimensional space of real vectors. In fact, concentrating on redundancies within patterns is a fairly new viewpoint. This point of view is in harmony with coding techniques where one designs codewords with a certain degree of redundancy and then use this redundancy to correct corrupted signals at the receiver's side.

We propose an online learning algorithm to learn the neural graph from examples and recall algorithms that use iterative message passing over the learned graph to eliminate noise during the recall phase. We gradually improve the proposed neural model to achieve the ability to correct a linear number of errors in the recall phase.

In the later stages of the thesis, we propose a simple trick to extend the model from linear to nonlinear regimes as well. Finally, we will also show how a neural network with noisy neurons–rather counter-intuitively–achieves a better performance in the recall phase.

Finally, it is worth mentioning that (almost) all the MATLAB codes that are used in conducting the simulations mentioned in this thesis are available online at `https://github.com/saloot/NeuralAssociativeMemory`.

Keywords: Neural associative memory, error correcting codes, codes on graphs, message passing, stochastic learning, dual-space method, graphical models

Résumé

Le milieu du 20^{eme} siècle a vu la publication de deux ouvrages pionniers dans les domaines de réseaux de neurones et la théorie des codes, respectivement le travail de McCulloch et Pitts dans 1943 et le travail de Shannon en 1948. Le premier a ouvert la voie à de réseaux de neurones artificiels et le dernier a introduit le concept de codage de canal, qui fait fiable communication sur les canaux bruyants possible.

Bien qu'apparemment lointain, ces domaines partagent certaines similitudes. Un exemple est la mémoire associative de neurones, ce qui est une classe particulière de réseaux de neurones capables d'mémorisation (apprentissage) d'un ensemble de motifs et de les rappeler ultérieurement en présence de bruit, soit, à extraire le motif mémorisé correcte à partir d'une version bruitée donné. En tant que tel, l'problème de mémoire associative de neurones est très similaire à celle qu'on rencontre en systèmes des communication où le but est de récupérer un ensemble de motifs (appelés mots de code) de manière fiable et efficace à partir de versions bruyants.

Plus intéressant encore, les techniques utilisées pour réaliser des mémoires associatives de neurones artificiels ressemblent beaucoup à certains des procédés de décodage utilisés dans les codes basés sur des graphes. Cela rend la phase de récupération de motifs dans les mémoires associatives de neurones très similaires à des techniques itératives de décodage dans la théorie du codage moderne.

Cependant, malgré la similitude des tâches et des techniques employées dans les deux pro-blèmes, il ya un écart énorme en termes d'efficacité. En utilisant de mots de code binaires de longueur n, on peut construire des codes qui sont capables de transmettre 2^{rn} mots de code de façon fiable sur un canal bruité, où $0 < r < 1$ est le taux de code. Dans les mémoires associatives neuronaux actuels, cependant, avec un réseau de taille n un ne peut mémoriser que $O(n)$ motifs binaires de longueur n. Pour être juste, ces réseaux sont capables de mémoriser un ensemble de motif qui sont choisis de façon aléatoire, tandis que les codes sont soigneusement construits. Néanmoins, cette généralité restreint sévèrement l'efficacité du réseau.

Dans cette thèse, nous nous concentrons sur la réduction de l'écart de performance entre les techniques de codage et de mémoires associatives de neurones artificiels en exploitant la structure inhérente à les motifs afin d'augmenter la capacité de récupération de motifs de $O(n)$ à $O(a^n)$, où $a > 1$. La figure 2 illustre l'idée derrière notre approche, à savoir, il est beaucoup plus facile à mémoriser d'autres motifs qui ont une certaine redondance comme des scènes naturelles dans le panneau de gauche que de mémoriser les motifs plus aléatoires dans le panneau de droite.

Figure 2: Lequel est le plus facile à mémoriser? Scènes naturelles de Van Gogh ou les peintures de cubisme de Picasso?

Plus précisément, nous nous concentrons sur la mémorisation de modèles qui forment un sous-espace (ou plus généralement, un variété). Le réseau de neurones proposé est capable d'apprendre et de manière fiable rappelant motifs donnés quand ils proviennent d'un sous-espace de dimension $k < n$ de l'espace à n dimensions des vecteurs réels. En fait, en se concentrant sur les redondance dedans des motifs est assez un nouveau point de vue. Ce point de vue est en harmonie avec les techniques de codage où l'on conçoit des mots de code avec un certain degré de redondance et ensuite utiliser cette redondance pour corriger les signaux corrompus à côté du récepteur.

Nous proposons un algorithme d'apprentissage "en ligne" pour apprendre le graphe de neurones à partir d'exemples et d'algorithmes de rappel qui utilisent itératif passage de messages sur le graphe appris à éliminer le bruit pendant la phase de rappel. Nous améliorons progressivement le modèle neuronal proposé pour atteindre la capacité de corriger un certain nombre d'erreurs linéaire dans la phase de rappel.

Dans les derniers stades de la thèse, nous proposons une truc simple d'étendre le modèle de linéaire à des régimes non-linéaires ainsi. Enfin, nous allons également montrer comment un réseau de neurones avec des neurones bruyants, plutôt contre-intuitivement, réalise une meilleure performance dans la phase de rappel.

Finalement, il faut mentionner que (presque) tous les codes MATLAB qui sont utilisés dans l'exécution des simulations mentionnées dans cette thèse sont disponibles en ligne à `https://github.com/saloot/NeuralAssociativeMemory`.

Mots-clés: Mémoire associative de neurones, codes correcteurs d'erreurs, les codes sur les graphes, le passage de messages, l'apprentissage stochastique, la méthode à espace dual, les modèles graphiques

Acknowledgements

For me, one of the most difficult jobs in the world has always been expressing gratitude to the extent I really mean it. For that, this part of the thesis has given me a headache for the past couple of days. I have been looking for the right sentences to convey my feelings and the deepest gratitude that I have for the people who helped me, in the journey that lead to this dissertation. I am not sure if I have been successful but since the deadline for submitting the thesis is approaching, here it goes.

To start, I would like to sincerely thank my thesis advisor, Prof. Amin Shokrollahi. I am truly honored to have the opportunity to work with Amin. In addition to his mathematical brilliance and expertise in many areas, such as coding theory, I learned a lot from his personality and him as a great *person*. I clearly remember the day that I met him for the first time, when I had come for my interview. At that time, all I knew about Amin was that he is quite famous for having invented some sort of error correcting code. However, his modest attitude really surprised me when I met him and although I thought is is going to be a stressful interview, I really enjoyed talking to him, something I still enjoy when we meet and chat about different things, from technical and research-related topics to football and politics.

I am also deeply thankful to Amin for kindly letting me choose my own research topic and offering me to work on applications of coding theory in biological systems when he, coincidentally, found out about my passion for bio-related topics. It was really joyful for me to work on something that I love and at the same time have the help of someone whose mathematical insight could solve a problem, that I had spent a couple of weeks working on, in a matter of minutes.

Another thing that I have learned from Amin is that, as a researcher, he is constantly looking for topics that are theoretically interesting and yet have very important *practical* applications. And for that matter, he is not at all afraid to enter totally new areas and learn completely new things (his new company, Kandu Bus, is a good case in point). He also does not worry about the end result and publishes new results when they are worth publishing. Overall, I am really thankful to him for having me here, for his constant support (both for me and my family) and for all the things I learned from him.

Next, I would like to thank ALGO's secretary, Natascha Fontana. From the first day that I arrived in Switzerland, she has been a constant source of support for me. Whether I had questions about administrative issues or needed help figuring out daily life issues in

Switzerland, I knew I could count on Natascha's kind and generous support. Thanks to Natascha, everything regarding our life as a graduate student went ever so smoothly. And above all, she has always been there for me as a source of emotional support, something I really appreciate and thank her for.

I would like to also thanks Giovanni Cangiani and Damir Laurenzi, the IT managers for ALGO and IPG, for their round-the-clock help with all sorts of issues that I encountered in using the computational infrastructure. Without a doubt, if it was not for their kind and prompt helps, most probably I could not have finished this thesis as a large portion of my work involves conducting extensive simulations for long periods of time.

Next, I would like to thank all those whom I had the privilege to work and collaborate with during these years. Starting with Raj K. Kumar, my friend, and office-mate for two years, with whom we started this project. Both of us were supposed to work on a different topic but very coincidentally discovered that we share a common interest and passion for bio-related applications. Without his insightful ideas and contribution, none of the accomplishments in this thesis would have become possible. Also, I would like to thank Prof. Wulfram Gerstner for his helpful comments and all the discussions we had with him and his group, to help us better understand the governing principles of neural networks. I would like to also deeply thank Amin Karbasi, for all the collaboration we had in the past two years, which has completely transformed the way I look at the problem I was working on. His broad and vast knowledge of many different areas, along with his seemingly endless energy, has been a great source of inspiration and technically crucial for this work. But above all, I cherish his friendship which makes working with him ever so enjoyable. I would like to also thank Lav R. Varshney, whom I have never met in person but have the opportunity to work with in the past couple of months. His bright ideas and expertise in both modern coding theory and neural networks was key to our progress on noisy neural associative memories and expanding our ideas to new areas. Finally, I am also thankful to Luoming Zhang, an ex-ALGO member, a friend and my first project supervisor here at EPFL. Despite my incompetence in programming and busy first semester at EPFL, he had been very kind and patient with me, helping me complete my first semester project and learn many things about applications of error correcting codes in magnetic storage systems. I would like to also thank Seyed Hamed Hassani and Vahid Aref, for their helpful comments and discussions on iterative decoding and spatially-coupled codes. I would like to also thank Masoud Alipour, for our numerous discussions on various applications of neural networks and learning systems, which I am sure will lead to even more fruitful collaborations in future. Finally, I am grateful to Hesam Setareh and Mohammad Javad Faraji, for all their kindness to help me better understand governing principles of neural networks and perform a sanity-check on the neural assumptions made in this work.

I am also thankful to the committee members who kindly accepted to take some time off their busy schedule near the end of the year to read my thesis, attend my defense and provide me with great comments regarding my work: Prof. Gerhard Kramer, Prof. Hans-Andrea Loeliger, Prof. Matthias Seeger, Prof. Wulfram Gerstner, the president of the jury, and Prof. Patrick Thiran, as an independent observer. In addition, I thank Prof. Bernard Moret and Prof. Wulfram Gerstner, the committee members in my candidacy exam, whose helpful comments and suggestions pushed me towards the right direction in my research as a PhD student.

I would like to also thank past ALGO members, together with whom I spent a wonderful time at EPFL: Masoud Alipour, with whom we shared the love of football and the DIY attitude, Ghid Maatouk for always being the source of energy in our lab, Omid Etesami for being a wonderful friend and showing me how a genius can be modest as well, Raj Kumar and Luoming Zhang, to whom I owe a lot in my research projects, Bertrand Meyer, for organizing lab's social activities, Ola Svensson for his always smiling, full-of-energy attitude and all his crazy ideas for creating startups, Yuval Cassuto for always sharing new things he just learned with the lab, and Mahdi Cheraghchi for his fantastic sharp sense of humor.

I would like to also acknowledge my friends, who have been absolutely supportive throughout these years and made life far away from home much less difficult: Vahid and Maryam and Dorsa, Omid and Nastaran, Amin and Marjan, Behrooz and Fatemeh, Hamed and Shirin, Masih and Maryam, Mohsen and Haleh, Ali and Maryam and Melika, Ali and Maryam and Kian, Reza and Mohammad, Farid, Saeed, Salman and Sharzad and Nikita, Arash and Mitra, Alireza and Sara, Mehdi (Jafari), Armin and Paris and Irin, Nasser and Sara, Sina and Nasibeh, Mani, Mohammad and Florin. I would like to also specially thank Reza Parhizkar, Seyed Hamed Hassani, Vahid Aref and Farid Movahedi Naeini for sheltering me during the first few weeks of my arrival in Switzerland. I am also thankful to my old pals from undergraduate studies, Ali, Hadi and Sina.

I am also thankful to the past and present members of the Information Processing Group (IPG) for the discussions we had from time to time and for all the fun we had together, specially playing foosball and football (in particular when we became the "disputed" champion of the I&C football tournament in 2012!).

I would also like to express deep gratitude to my previous supervisors, from whom I learned a lot both on a personal and professional level: Prof. Mohammad Reza Aref, Prof. Mohammad Hasan Bastani, Prof. Babak Hossein Khalaj, Prof. Mohammad Reza Pakravan and Prof. Mehdi Sadeghi.

My research work was supported by Grant 228021-ECCSciEng of the European Research Council, which I gratefully acknowledged. I would like to also thank EPFL (and all administrative staff), for giving me the opportunity and facilitate my studies as a doctoral students here.

And very special thanks to...

...my "family" for all their unconditional love and support throughout my life. Although there are no words that could express my gratitude, I would like to thank my mom, Nooshin Roostan, for all her love and care, for always making sure that I am on the right track, for teaching me the value of knowledge and the importance of reading by taking me to the library at the other end of the city every other week since the age of 4. And above all, for all the sacrifices she made so that I could focus on and enjoy studying in a warm and comfortable environment at home.

I would like to express my heartfelt gratitude to my dad, Ahmad Salavati, for all his support and love, for his encouragements to continue my studies until the last stage, and for teaching me that there is a time for doing anything, and the time for studying is when you are younger. But more than anything, I would like to thank him for being a solid wall, absorbing all the pressures of the outside world so that we enjoy a relaxed environment at home, not knowing how challenging life can be from time to time.

I would like to sincerely thank my brother, Ehsan, for all the fun and joy we had growing up together, for helping me significantly the year I was studying to enter the university (by not allowing me touch the computer to play games!) and for all the things I have learned from him. It is really good to have someone like him around since, despite being younger than me, he has been a role model for me on how one could balance work and personal life and be good at both.

I am also deeply thankful to my mother and father in law, Nassrin Jafari and Saeed Ashari Astani, for their constant support, energy, love and encouragement throughout these years, which makes me a very lucky person to receive so much kindness and love from their part. But more than anything, I am genuinely grateful to them for treating me like their own son. This means a lot to me and I am really indebted to them for that. I would like to also thank my brother in law, Ramtin, who was like a brother to me in the first place and before becoming my brother in law. I am thankful to him for all the fun we have together, for his good taste of movies (not so good with music though!), and for always making me feel very important when we have a discussion from time to time.

And the last but not the least, my deep and sincere thanks go to my wife, Negar, for all her love and support, for all her sacrifices she made so that I worry about nothing but studying, for believing in me even during the times when I did not believe in myself and for always reminding me how lucky I am, as whenever I feel terribly concerned about something, I would just look around, see her by my side and realize that I have *all* that matters in life right beside me. Everything else, including a PhD degree, is just an optional extra. Having such an angel by my side makes me the luckiest man on the face of earth.

I am thankful to my "family", simply for the fact that my success makes them even more happier than me.

Contents

Part I

Preliminaries

Chapter 1

Introduction

I am really bad at typing. Without the help of an automatic spell-checker, anything that I write, including this thesis, would most certainly contain many typos, like this onee! Yet, both you and I could easily detect and correct many misspelled words, and in many circumstances even uncounsiously (well, unconsciously, thanks to the spell-checker!).

This ability of our brains, among many others, is truly staggering, especially for someone whose background is in designing coding techniques to deal with the problem of noise in communication channels. There, we face the same problem, since what the receiver receives is not exactly what the transmitter has transmitted, due to the noise in the channel. Thus, we must find a way to infer what the transmitter had in mind from a corrupted version that we have at hand.

The fact that we enjoy using our cellphones or laptops to receive calls or download files in a noisy environment such as the wireless medium means that we have been successful in designing such coding techniques. However, finding how such techniques relate to their equivalent in neuronal networks (e.g. our brain) is certainly worth more investigations.

This is how the project that lead to this thesis was initiated. Fascinated by these similarities in the objectives, and motivated by recent advances in applying neural networks to the design of better coding methods [1,2], we set out to study the reverse problem: that of using theoretical methods in coding theorey to better understand neural networks.

We encountered many examples that were similar in nature to communication over a noisy channel [3,4]. In fact, the neural medium in the nervous system is a noisy environment and the messages neurons exchange among each other is susceptible to noise [5]. So it would be interesting to see how a system that is built from "unreliable" components could perform such delicate and accurate tasks as our nervous system is capable of. One immediate guess would be to check if there are special coding techniques performed by neurons to deal with the communication noise over the neural channel [3, 4].

Nevertheless, the *internal* noise is not the only source of uncertainty in the nervous system as it should also be able to deal with *external* sources of noise. There are numerous situations where the system should make the correct decision from corrupted and partial information. The misspelled words example which we mentioned earlier is a very good case in point. Another good example is furnished by *neural associative memories*, which will be the main focus of this thesis.

1.1 Neural Associative Memories

Briefly speaking, neural associative memories are a particular class of neural networks capable of memorizing a given set of patterns and recalling them later from corrupted/partial cues. Therefore, an associative memory has a learning and a recall phase. In the learning phase, the proposed approach determines the connectivity matrix of the weighted neural graph from the given patterns. The learning is performed in such a way that the memorized patterns are the stable states of the system, meaning that the network does not converge to a different pattern once initialized with a memorized pattern.[1]

During the recall phase, we are given a noisy version of a memorized pattern, where certain entries are missing or are corrupted due to noise. The neural system should then find and retrieve the correct pattern from this partial cue, utilizing the connections in the neural graph that has captured information about the memorized patterns in the learning phase.

Since neural associative memories also involve dealing with noise and corruptions in the retrieval phase, they are close to what coding techniques attempt to achieve in communication systems, namely, to eliminate the effect of noise in the communication channel to retrieve the correct "codeword" from a corrupted received version.

More interestingly, both methods use similar techniques to accomplish a similar task: in neural associative memories we have a (neural) graph and a set of update rules that dictate the message passing process which is responsible to eliminate noise during the recall phase; and in modern coding techniques, such as LDPC or Raptor codes, we have a graph which is accompanied by a message passing process to eliminate the effect of channel noise and to yield the correct transmitted pattern.

[1]To be more precise, this model describes an *auto-associative* memory. In *hetero-associative* memories we have virtually the same concept, except now we memorize the pair-wise relation between two patterns of different length, e.g. the name of an object and its image.

1.2 Problem Formulation

Despite the similarity in the task and techniques employed in both problems, there is a huge gap in terms of efficiency. Using codewords of length n, one can construct codes that are capable of reliably transmitting 2^{rn} *structured* codewords over a noisy channel, where $0 < r < 1$ is the code rate. In current neural associative memories, however, with a network of size n one can only memorize $O(n)$ *random* patterns of length n.

Bridging this efficiency gap is the main focus of this thesis. More specifically, we are interested in designing a neural network which is capable of

1. Learning a set of C patterns (vectors) of length n in an "online" and gradual manner, i.e., being able to learn from examples.

2. Correcting a linear (in n) number of errors during the pattern retrieval (recall) phase.

3. Ensuring that the pattern retrieval capacity is exponential, i.e. $C \propto a^n$, for some $a > 1$.

1.3 Solution Idea and Overview

To achieve these properties, we will borrow ideas from statistical learning and coding techniques to accomplish the first two properties. To make exponential pattern retrieval capacities possible, we focus on memorizing patterns with suitable regularities and structures. To expand this concept, note that the mainstream work on neural associative memories requires the network to be able to memorize any set of *randomly* chosen patterns of length n. This requirement surely gives the proposed model a certain sense of generality. Nevertheless, this generality seems to be severely restricting the efficiency of the network, since a similar model that only concentrates on *structured* patterns can achieve much higher retrieval capacities in modern coding techniques. In fact, it has been shown that *any* method for choosing the $n \times (n-1)$ neural weights will result in a neural associative memory that can not memorize more than $2n$ random patterns [6].

Furthermore, real-world scenarios seem to support this idea as well, as illustrated in Figure 1.1: it is much easier to memorize more patterns that have some regularity and redundancy, as in natural scenes shown in the left panel, than to memorize (seemingly) random images, like the one in the right panel. In addition, dealing with any corruption in the left image is certainly much easier than in the one on the right.

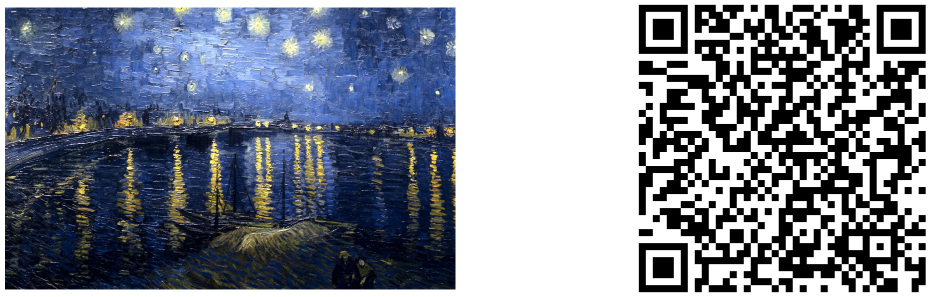


Figure 1.1: Which one is easier to memorize?

Therefore, our goal in this thesis would be to propose a neural network which is capable of memorizing exponentially many patterns that contain suitable regularities. We propose an online learning algorithm to learn the neural graph from examples and recall algorithms that use iterative message passing over the learned graph to eliminate noise during the recall phase.

We start with a failure: in Chapter 4 we explain our first attempt based on memorizing patterns that have low correlation to each other improves the capacity but fails to achieve the exponential benchmark. Thus, we move to a different model and in Chapter 5 we focus on the patterns that form a subspace. This change in the strategy pays off as we could achieve exponential pattern retrieval capacities. We gradually improve the proposed neural model through Chapters 6 to 8 to achieve the ability to correct a linear number of errors in the recall phase as well.

In the later stages of the thesis, we propose a simple trick to extend the model from linear to nonlinear regimes, i.e. instead of considering only the patterns that come from a subspace, we also consider those that form a manifold. Chapter 9 will be dedicated to describing this idea in more details.

Finally, in Chapter 10 we will make a practical modification to the proposed model by making the neurons in the proposed model noisy as well. This is closer the real neurons, where the firing rate is a random number sampled from a distribution whose mean and variance depend on the neurons' inputs. Rather surprisingly and counter-intuitively, we show that this modification actually improves the network and we could achieve a better performance in the recall phase.

Throughout the thesis, it is assumed that the reader is familiar with the basic concepts of neural networks and coding techniques. If that is not the case, a short introduction on either topics is provided in Chapter 2 for the interested reader.

1.4 Our Model

The models we are going to use in this thesis all inherit basic properties of artificial neural networks, namely, each neuron calculates a weighted sum over the messages received from its neighbors and (possibly) applies a non-linear function to update its state and send information to its neighbors (for a short introduction on principles of neural networks see Chapter 2).

However, there are some details that distinguish our model from other models in the mainstream work on neural associative memories.[2] More specifically, the key properties of the model used in this thesis are

1. The patterns we are going to memorize contain some sort of regularity. This regularity can be of the form of having low correlation with each other or belonging to a subspace or a manifold (of the space of all possible patterns). This is the keypoint in this thesis and makes our work different from the mainstream approaches in designing associative memories where there are no restrictions on the patterns.

2. The state of neurons are integers from a finite set of non-negative values $\mathcal{Q} = \{0, 1, \ldots, Q-1\}$. Note that in general, $Q > 2$ and, thus, our model is different from the binary neural models for such cases. The integer-valued states could be interpreted as the short-term (possibly quantized) firing rate of neurons.

3. The neural graph is bipartite (except for the network in Chapter 4).

4. Since we work with integer-valued neurons, the noise in the recall phase is also integer-valued (in the range of $\{-S, \ldots, S\}$, for some $S > 0$). This noise can be interpreted as a neuron missing a spike or firing more spikes than it should. Nevertheless, note that the same noise model can easily be extended to contain "erasures" in the data (loss of information) as well. For instance, if a position is erased, we can treat it as a 0, which is equivalent to having a noise value that has the same magnitude as the correct symbol but with the opposite sign (although one could benefit from knowing the *position* of erasures to design more efficient algorithms).

The above properties make the proposed model very similar to the graphical model used to decode codes on graphs, such as LDPC codes. As we will see later, the algorithms proposed in this thesis look very similar to message passing techniques to decode such codes. However, there are two significant differences between the model and proposed algorithm here and the standard Belief Propagation (BP) technique used to decode LDPC codes

[2]Note that despite these differences, our model is still a fall within the domain of artificial neural networks, and as such, can be implemented by relevant algorithms.

1. The neurons can not transmit different messages over different outgoing links. Everything that a neuron transmits goes to all its neighbors.

2. A neuron can not have access to the individual messages received over its links from its neighbors. The only quantity that is available to a neuron during the decision-making process is a weighted sum over the received messages.

Both these differences are imposed by the simple nature of neurons and make it difficult to apply exactly the same techniques (like BP) to perform the recall phase in a neural associative memory.

Why this model?

The considered model is appealing in several senses. First of all, the "regularity" assumption in the patterns makes it possible to design efficient associative memories, as will be seen later on. Secondly, the non-binary neural model means that it is possible to have exponentially many integer-valued patterns in the dataset that also form a subspace or manifold. This makes it possible to have an exponential pattern retrieval capacity.[3]

Furthermore, the non-binary assumption also enables our model to integrate both *rate codes* and *time codes* in a real neuronal network (see Chapter 2 for the definition of rate and time codes in neurons). More specifically, the integer-valued state of a neuron could be considered as its short term firing rate (rate code) or if we divide the time interval into very small "bins", the binary expansion of the same quantity can be considered as the firing pattern transmitted by the neuron in each bin (for instance in a time interval of 3 bins, $Q = 7$ and the pattern 100 can be represented by 4 as the state of the neuron). This is particularly pleasing because it makes future refinements easier and also help us to have a more biologically-relevant model.

1.5 Where Might This Thesis Be Useful?

The results provided in this thesis are mostly theoretical. However, there are numerous practical applications where such theory could help. For instance, due to their structure and capability to retrieve patterns from partial information, associative memories have natural applications in content-addressable memories [7] as well as search engine algorithms that use not only users' inputs but also the association between the keywords in the search domain (see [8] for example). The inefficiency of current neural associative memories in reliably

[3]Note that we might not be able to get similar pattern retrieval capacities in a binary model with our setting since there might not be exponentially many binary patterns in a subspace of $0, 1^n$.

memorizing a large number of patterns acts as a barrier in deploying these algorithms in large-scale practical systems. By proposing a method to increase the pattern retrieval capacity, this thesis might make it one step closer to widespread adoption in practical systems.

Furthermore, and as mentioned earlier, the algorithms proposed in this thesis have close relations to those performed by codes on graphs (e.g. LDPC and Raptor codes). Chapter 10 provides a surprising result that might be of interest for practical graph-based decoding techniques, especially when the length of the codewords is limited.

1.6 Some Final Preliminary Remarks

Before moving to the technical parts of the thesis, I would like to emphasize one point (again), as it is really close to my heart: the project that lead to this thesis was initiated to explore different applications of coding techniques (and the theory behind them) in relevant biological systems, especially artificial neural networks. I hope that the examples mentioned in the beginning of this introductory chapter have convinced the reader that the results given in this thesis are just scratching the surface and there are numerous other potential applications. This belief is supported by the fact that there is a rich theoretical background on graphical models and algebraic systems in coding theory which could be useful in analyzing (and designing artificial) neural networks as they have graphical models deep in their core.[4]

[4]For the sake of completeness, one could also think of other biological systems that deal with graphs, for instance Gene Regulatory Networks, where similar arguments apply.

Chapter 2

A Short Introduction to Neural Networks

This chapter is dedicated to very briefly introducing the main concepts of neuronal and neural networks. We start by providing a short description of the anatomy of a neuron, how neurons communicate via electrical pulses and the way a neuron encodes information based on the received input from its neighbors.

Then, we introduce the artificial neuron model, discuss the way it encodes information and introduce the model we will be using in this thesis.

2.1 Anatomy of a Neuron

Neuronal systems are made up of small cells called neurons. Each neuron is composed of four main parts, as shown in Figure 2.1:

1. Soma, which is the main cell body,
2. Axon, which carries neural messages (called action potential) towards the neighboring neurons,
3. Synapses, where an electrical signal is transformed into chemical form and
4. Dendrites, which re-transform chemical signals back into electrical format and transmit them to the soma.

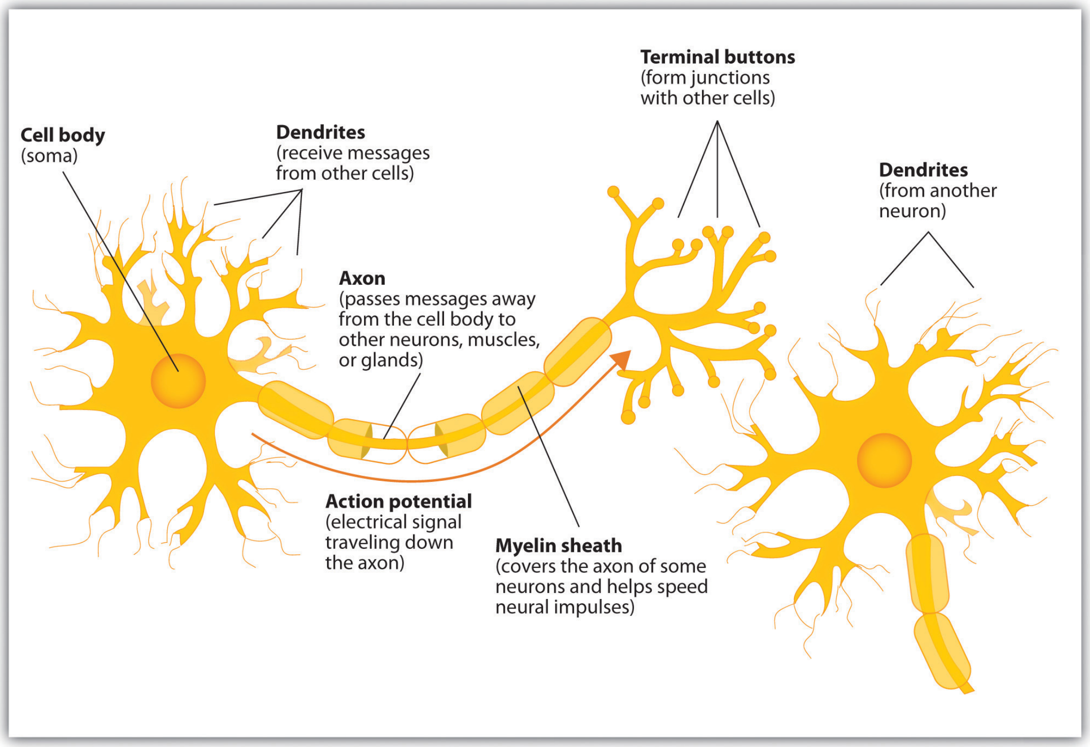

Terminal buttons
(form junctions with other cells)
Cell body
(soma)
Dendrites
(receive messages from other cells)
Dendrites
(from another neuron)
Axon
(passes messages away from the cell body to other neurons, muscles, or glands)
Action potential
(electrical signal traveling down the axon)
Myelin sheath
(covers the axon of some neurons and helps speed neural impulses)

Figure 2.1: Anatomy of a neuron [9]

The cell body or soma of a typical cortical neuron ranges in diameter from about 10 to 50 μm [10]. Length of dendrites vary from a few microns up to 100 microns. In contrast, axons are much longer and a single axon could even traverse the whole body.

The soma receives signals from dendrites and transmits another signal based on the received input along the axon. The structure of a dendrite, which contains many branches, allows a neuron to receive signals from many other neurons via synapses. The axon makes an average of 180 synaptic connections with other neurons per mm of length while the dendritic tree receives, on average, 2 synaptic inputs per μm [10].

However, the most important feature of neurons is their specialty in transmitting electrical pulses. These pulses, usually called *action potentials* or *spikes*, makes neural message passing and information processing possible.

Action Potential

Briefly speaking, an action potential is generated as a result of chemical reaction at a given neuron. Each neuron has many ion channels that allow ions, predominantly sodium (Na^+), potassium (K^+), calcium (Ca^{2+}), and chloride (Cl^-), to move into and out of the cell. This transform in ion concentration level creates a potential difference, which propagates throughout the cell as an electric pulse(from soma via axon or from dendrites towards soma). In synapses, the action potential results in the release of chemicals known as *neurotransmitters*, which in return triggers (or inhibits) an action potential in neighboring neurons.[1]

The shape of an action potential is shown in Figure 2.2. It is an electrical pulse with amplitude of almost $100mV$ and approximate width of $1ms$.

Due to structure of the neuron and ion channels, it is impossible to generate an action potential right after another. One must wait a certain amount of time before the neuron is able to generate the next action potential. This period is called the *absolute refractory period*. Moreover, for a longer interval after generation of an action potential, producing another action potential is more difficult. This longer interval is called *relative refractory period*.

Action potentials traverse along the axon in an active process, meaning that the ion currents are generated continuously along the way through the axon membrane. This prevents an action potential to become severely attenuated (and vanishing eventually). Nevertheless, in a particular class of neurons, where there is *myelin* sheath around the axon, spikes could travel along the axon without being regenerated in distances up to $1mm$. Then, action potentials are regenerated in openings in the myelin sheath called *Ranvier nodes*. This process is known as *saltatory transmission* which is much faster and resembles the transmission of electrical signals along power transmission lines.

Axons terminate at synapses where the electrical pulse opens ion channels producing an influx of Ca^{2+} which leads to the release of a neurotransmitter. The neurotransmitter binds to the dendrites of the neighboring neuron(s) causing ion-conducting channels to open [10]. Depending on the nature of the ion flow, the synapses can either be excitatory, where it contributes to triggering another action potential, or inhibitory, where it tries to inhibit the neighboring neurons to generate an action potential.

[1]More specifically, under resting conditions, the potential inside the cell membrane of a neuron is about $-70mV$ relative to that of the surrounding medium. In this case, the cell is said to be *polarized*. When positively charged ions flow out of the cell (or negative ions flow inward), a current is created which makes the membrane potential more negative. This is called *hyperpolarization*. The reverse process, in which positive ions move inward, makes the membrane potential less negative (or even positive), a process called *depolarization*. If a neuron is depolarized sufficiently to raise the membrane potential above a threshold level, the neuron generates an action potential [10].

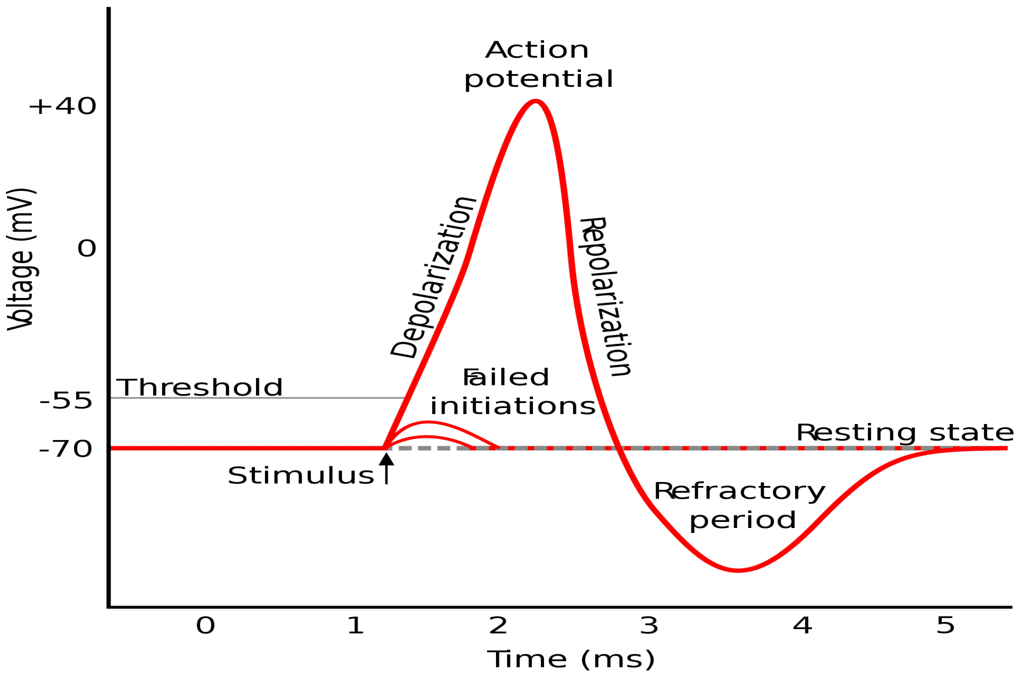


Figure 2.2: The schematic of an action potential (from Wikimedia)

Rate Coding, Temporal Coding

A single action potential could carry a relatively small amount of information, e.g., signaling the existence or lack of a stimulus. Nevertheless, a train of action potentials, usually called a spike train, could be more informative as the group of action potentials provide more granular information regarding stimulus or the output of previous processing stages.

However, there is not a universal agreement on how exactly spike trains encode information. In certain cases, such as sensory neurons, what seems to be important is the number of spikes in a short time frame, rather than their relative timing. In fact, it seems that rate coding is the main way of encoding information in the majority of cases [11].

On the other hand, there is *temporal coding* where the relative timing of spikes play an important role in encoding information. Although this makes the system extremely sensitive and prone to errors, there are pieces of evidence that in vital situations where there is not enough time for calculating a short-term average on the number of spikes (such as escaping from a nearby threat), spike timing plays an important role in the decision making process

[11].

2.2 Neuronal Networks

A neuron on its own does not have extensive information processing capabilities. However, when neurons are connected, their computational power grows enormously. A group of inter-connected neurons constitutes a network. There are two important factors that affect the properties of such networks: the number of synapses between the neurons and the weights of these synapses. By appropriately adjusting these two parameters, one can obtain different networks performing various tasks.

A neural network can then be characterized be a connectivity matrix W that specifies the connection weights between any pair of neurons. More specifically, the entry W_{ij} represents the weight of the connection from neuron i to neuron j. If $W_{ij} > 0$, the connection is excitatory. A negative weight represents an inhibitory connection and a 0 means the two neurons are not connected. Note that the matrix need not be symmetric (and in fact it often is not).

Network dynamics: the firing rate model

Neurons interact by exchanging electrical messages over this network. There are different models to capture the influence of neurons over their neighbors in the neural network. One simple yet intuitive approach is to focus on the rate coding and derive how the firing rate of pre-synaptic neurons affect that of the post-synaptic one [10]. To this end, let I_s and v be electrical current to soma and the firing rate of the post-synaptic neuron, respectively. Furthermore, let u_i for $i = 1, \dots, n$ be the firing rates of the n pre-synaptic neurons that are connected to the post-synaptic neurons with a synaptic efficacy (weight) of w_i. Figure 2.3 illustrates the model.

Now if a spike from a pre-synaptic neuron j arrives at time t_i, it contributes to I_s according $w_j * K_s(t)$, where K_s is the synaptic kernel [10] (for simplicity, we consider the same kernel for all synapses). Assuming that the spikes at a single synapse are independent, the total contribution of the pre-synaptic neuron x to I_s is:

$$w_j \sum_{t_i < t} K_s(t - t_i) = w_j \int_{-\infty}^{t} K_s(t - \tau) u_j(\tau) d\tau, \tag{2.1}$$

where $u_j(\tau)$ is the firing rate of neuron j at time τ.[2]

[2]Note that this equation is an approximation that holds only when neurons have many pre-synaptic connections with uncorrelated firing responses.

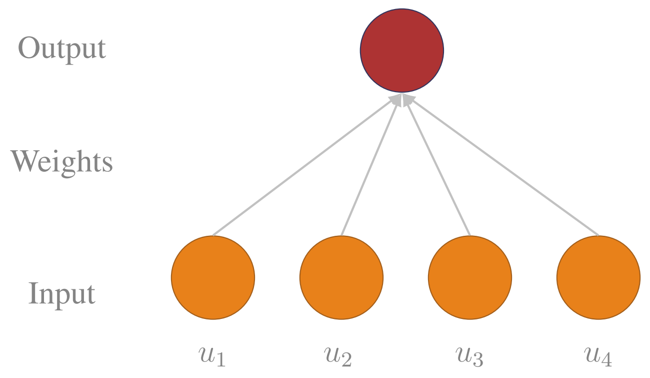


Figure 2.3: Firing rate model for a simple neural network

Therefore, by summing up the contribution of all pre-synaptic neurons we obtain

$$I_s = \sum_{j=1}^{n} w_j \int_{-\infty}^{t} K_s(t-\tau) u_j(\tau) d\tau. \tag{2.2}$$

We can rewrite the above equation as the differential equation

$$\tau_s \frac{dI_s}{dt} = -I_s + \sum_{j=1}^{n} w_j u_j. \tag{2.3}$$

Finally, we can estimate the output firing rate, v, as a function of I_s, i.e. $v = F(I_s)$. Different choices for $F(.)$ can be considered, e.g., the sigmoid function or a linear-threshold function $F(I_s) = [I_s - \theta]_+$ where θ is the firing threshold.

Hence, the output of the network is governed by the dot product of weight vector w and the input firing rates, u. In fact, this is a crucial property of neuronal networks since by adjusting w, they can perform different dynamical behaviors which help them to accomplish different tasks including memorizing, learning and pattern recognition.

Network topology

Based on the received (weighted) integration of signals, neuron send proper responses to their neighbors. Thus, an important factor that contributes significantly to accomplishing a specific information processing task is the network topology. In cortex and other areas of the brain, we find three major interconnection types [10]:

- Feed-forward: these connections bring signals from an earlier stage and possibly process it along the way.
- Recurrent: recurrent synapses connect neurons in a particular processing stage.
- Top-down: these interconnections bring a signal back from later processing stage.

What makes neural networks quite unique is their ability to adjust the network topology and connection weights according to the task at hand. This process is usually referred to as learning.

Learning and Plasticity

Neural networks are *plastic* in the sense that they can *learn* what to do and how to do it. In this context, the connection weights in a neural graph are adjusted such that a particular relationship is constructed between the input and output of the network. Depending on the circumstances, a learning algorithm can be

1. Supervised: in supervised learning, we give both input and required (correct) output to the network. Hence, the network knows the answer in advance. All the network has to do is to choose its synapse weights such that the specified input results in the correct output.
2. Unsupervised: in this case, the explicit relationship between input and output is not given. Instead, a cost function is minimized during the learning process which in the end specifies the input-output relationship.

Both cases usually involve an iterative process, where in each iteration a sample is randomly selected from input data and the neural weights are slightly adjusted in the proper direction.

2.3 Artificial Neural Networks

McCulloch and Pitts [12] introduced a simple model of neurons. In their proposed model, a neuron is modeled as a threshold device that computes a weighted sum of received input messages and yield a zero or one, according to how this weighted sum compares to a specified threshold.

More generally, an artificial neuron is modeled as a device that receives a weighted input sum and applies a nonlinear transfer function to compute its output state. The nonlinear mapping, often called *activation function*, is given by the equation below [13]:

$$v = g(\sum_j w_j u_j - \theta), \tag{2.4}$$

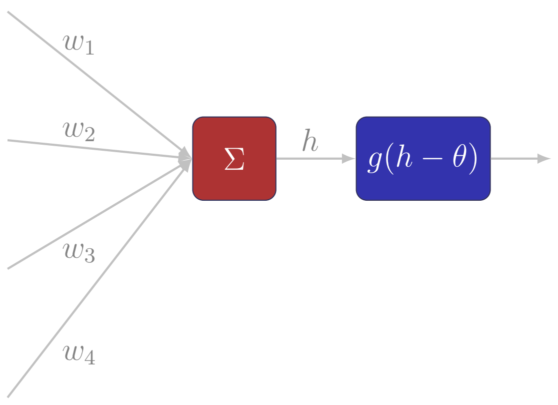


Figure 2.4: Artificial neuron model [13]

where v is the output state of the neuron, w_j is the connection weight from neuron j to the given neuron, θ is a fixed threshold and $g(.)$ is the nonlinear activation function. Figure 2.4 illustrates the model. If $g(x) = \text{sign}(x)$, we obtain the McCulloch-Pitts model.

Despite its simplicity, the artificial neuron is capable of performing powerful computational tasks. More specifically, McCulloch and Pitts proved that a synchronous assembly of binary neurons is capable of universal computation for suitably chosen weights.

Nevertheless, the artificial neuron differs from the real one in certain key aspects [13]:

1. Real neurons are often not threshold devices and act more in a continuous way.

2. Many real neurons perform a *nonlinear* summation over their inputs. There can even be significant logical process, such as AND, OR or NOT operations within the dendritic tree.

3. Real neurons do not a have a fixed delay and they are usually not updated synchronously.

4. Real neurons are noisy in the sense that a fixed input does not result in the same output *all* the time but rather on *average*.

5. In a real neural network, neurons typically emit only one kind of neurotransmitter [6]. This phenomenon, known as Dale's law, makes a neuron, and all its outgoing synapses,

either excitatory or inhibitory. In our terminology, this means that the outgoing weights of a neuron should all be negative or positive.

Throughout this thesis, we will work with the standard artificial neuron model. However, in the last chapter we will show how a noisy neural model will in fact improve the performance of the proposed algorithms. Furthermore, for certain algorithms in this thesis, the third point would in fact be more desirable as it makes *asynchronous* updates more sensible. Finally, for other applications the second limitation above can be alleviated by considering several artificial neurons to represent a single unit that mimics the behavior of a real neuron.

2.4 Our Neural Model

As mentioned earlier, the neural model considered in this thesis is based on the standard artificial neural model, given by equation (2.4). As a result, a neuron could only calculate a linear summation over its inputs and apply a possibly nonlinear function to return the output. However, with the exception of Chapter 4, in all chapters we add the restriction that the output is a non-binary, non-negative integer (in Chapter 4 the neurons are binary as in the McCulloch-Pitts model [12]). The integer-valued states of neurons can be interpreted as their short term (possibly quantized) firing rate. In all cases, the neurons are deterministic. However, in the last chapter we consider a noisy neural model and show how it can positively affect the obtained results.

Furthermore, during the recall phase of all proposed neural architectures, we consider the retrieval of a pattern from a corrupted (or partial) cue. The corruption is modeled by a noise vector with integer-valued entries. More specifically, and to simplify the analysis, in most cases the noise is modeled as a vector whose entries are set to ± 1 with some probability p_e and 0 otherwise, where p_e shows the probability of a symbol error. This model can be thought of as neurons skipping a spike or firing one more spike than they are supposed to. Nevertheless, unless mentioned otherwise, the algorithms could work with non-binary integer noise values as well, i.e., where each entry in the noise vector is chosen independently and is a non-zero integer from the range $\{-S, \dots, S\}$ (for some integer $S > 0$) with probability p_e and 0 with probability $1 - p_e$.

Other noise models, such as real-valued noise, can be considered as well. However, the thresholding function $f : \mathbb{R} \rightarrow \mathcal{Q}$ will eventually lead to integer-valued "equivalent" noise in our system.

Finally, it is not hard to extend the algorithms to deal with erasures (partial loss of information) as well. One naive approach could be to model an erasure at neuron x_i as an integer noise with the negative value of x_i. So once we have established the performance of our algorithm for integer-valued noise, it would be straightforward to extend the results as a lower bound on the performance of the algorithms in the presence of erasure noise models, because in that case one could take into account the known position of errors to achieve a better performance.

To summarize, in our neural model

1. Artificial neurons calculate a linear summation over their input link and apply a (possibly) nonlinear function to update their states according to equation (2.4).
2. The states of neurons are non-negative and bounded integer values, from a set $\mathcal{Q} = \{0, \dots, Q-1\}$ for some $Q \in \mathbb{Z}^+$.
3. The neural graph is weighted with real-valued weights.
4. In the recall phase, each entry in the pattern vector is corrupted due to noise i.i.d. at random with some probability p_e. The noise affecting each entry is an additive integer, drawn uniformly at random from the set $\{-S, -(S-1), \cdots -1, 1, \dots, S-1, S\}$ for some $S \in \mathbb{N}$. For simplicity, in many cases we assume $S = 1$.

Chapter 3

A Short Introduction to Coding Theory

This chapter serves as a brief explanation of main concepts in coding theory. It is by no means meant to go into details and solely introduces the *ideas* that are used to design efficient methods to deal with noise in communication channels. These ideas are crucial and inspiring for our work, as we will utilize some of them to achieve exponential pattern retrieval capacity in neural associative memories.

3.1 Communication Over Noisy Channels: the Role of Redundancy

Communication channels are often noisy. Figure 3.1 illustrates a widely-used model for a scenario where noise is additive. In such circumstances, what the receiver receives could be different from what the transmitter has transmitted. The question is if it is possible to tell if what the receiver has received is corrupted by noise in the first place and if it is possible retrieve the transmitted message from this corrupted version. Designing efficient methods that accomplish one or both objectives is among the main tasks we are faced in coding theory. Such methods are usually called *channel coding* techniques.

Methods that deal with channel noise are divided into two major categories: automatic repeat-request (ARQ) and forward error correction (FEC). In the first approach, we employ algorithms that can only tell if the received message is corrupted due to noise. In that case, the receiver asks the transmitter to repeat the message. In the second method, if the noise is fairly limited the algorithm is capable of retrieving the correct message without asking the

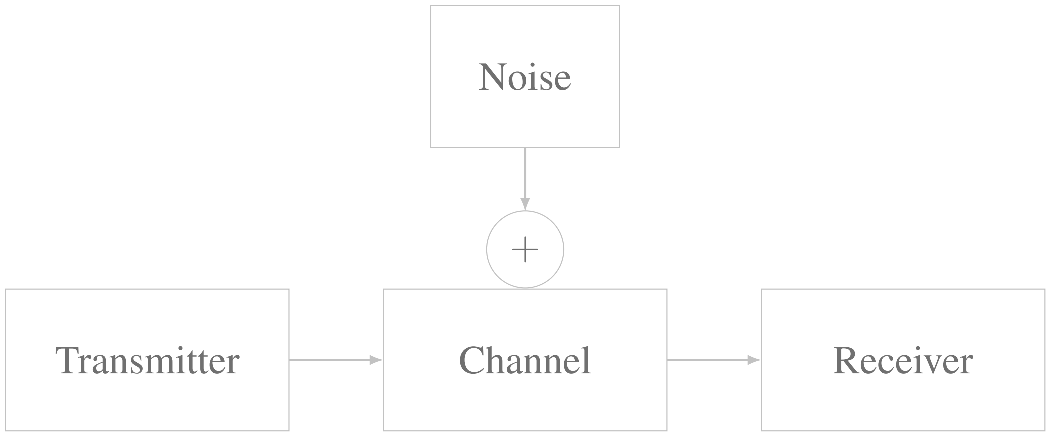


Figure 3.1: Noisy communication channel

transmitter to resend the information. In this thesis, we are more interested in algorithms of the second type.

Both ARQ and FEC approaches rely on a simple trick to accomplish their respective objectives: add some redundancy to the original message before transmitting it. The additional redundancy will then come in handy at the receiver's side to establish whether the received message is corrupted and to guess its correct content.

As a simple example to see how this idea works, consider a binary channel where we are interested in transmitting a single bit $x = 0/1$. Furthermore, and due to the noise, the output of the channel would be different from the transmitted bit with some probability $0 < p < 1$. Additionally, we assume the noise to act independently on each transmitted bit. Without any coding technique whatsoever, the receiver has no way of telling if a received bit is equal to the original one sent by the transmitter. Nevertheless, we could employ a simple technique to reduce the probability of making an error significantly: we repeat each bit n times. Thus, instead of sending 1, we will send the vector 111 for $n = 3$. At the receiver, we will examine each received message to see if all n entries are equal. If so, we accept it as the correct transmitted piece of information. Otherwise, we will for instance ask the transmitter to resend the message. This simple trick will reduce the probability of making an error from p to p^n. More sophisticated techniques could be used to enable the receive to also retrieve the correct message on its on without asking for a re-transmission.

At this point, it is hopefully clear that adding redundancy will help us to increase the reliability of communication in noisy environments. However, it did come at a price: waste of resources. Adding redundancy means sending more bits (symbols) over the channel which translates to spending more energy, time and computational resources. Thus, it would be natural to ask what the best balance would be that makes it possible to have reliable communication (i.e., with probability of error tending to zero) and minimize the waste of resources at the same time. Shannon answered this question in 1948 [14] with the notion of *channel capacity*, which specifies the best trade-off between close-to-absolute reliability and least amount of redundancy added to the transmitted messages.

3.2 Channel Capacity

Consider the model shown in Figure 3.1 and for simplicity assume we are interested in transmitting messages that are binary vectors of length k. In order to deal with channel noise further assume that we somehow *encode* the message by adding some sort of redundancy and consequently increasing its length to $n > k$. We call the resulting vector of length n a *codeword*. Finally, define the *code rate* r as $r = k/n$, i.e., the ratio of "useful" bits to the total amount of transmitted bits.

Then, the "noisy channel coding theorem" states that for each noisy channel, there is an upper bound c on the code rate, such that if $r < c$, there exist codes that allow arbitrarily small probabilities of error at the receiver's side. Furthermore, if $r > c$, *all* coding schemes will result in a probability of error that is greater than a minimal level, which increases as rate grows [14].

The quantity c is often called the *channel capacity*. In many cases, it can be calculate from the physical properties of the channel.

The noisy channel coding theorem is important for different reasons:

1. It shows that reliable communication in noisy environments is indeed possible, a fact that had not been know before.

2. It provides a bound on the minimum amount of redundancy required to achieve reliable communication.

However, the theorem only shows the *existence* of coding techniques that could achieve reliable communication. It does not actually give an efficient method to accomplish this objective. As a result, coding theorists have put a great deal of effort into finding appropriate coding techniques whose rate is arbitrarily close to channel capacity while achieving reliable communication over the noisy channel. While Forney had proposed one such coding schemes [15], it was only recently that the relationship between the running time of the decoder and the proximity to the channel capacity has been extablished [16].

3.3 Linear Coding Strategies

As Shannon did not describe a particular channel coding method in his theorem, different strategies could be used to add redundancy to the transmitted codewords. The only common part is that the messages of length k should be somehow mapped to a larger space of dimension n, i.e., codewords of length n. However, among different strategies, the one that is based on a linear mapping between the messages and the codewords has been extensively considered in the past 60 years due to its computational simplicity, which makes it more suitable for practical applications.

To simplify the argument, suppose we are interested in communication over a binary channel, and, therefore, are interested in binary messages and codewords. Furthermore, assume that all operations are performed in the binary field, $GF(2)$. In linear codes, one is faced with the task of finding a suitable *linear* mapping between the space of messages with dimension k, to the space of codewords with dimension $n > k$, while $r = k/n < c$. As a result, it is not hard to see that in the final n-dimensional space, the codewords actually form a subspace of dimension k.

The fact that codewords form a subspace in the n-dimensional space provides the first stepping stone in dealing with noise in communication channels. Let $H_{n-k\times n}$, called the *parity check matrix* be the binary matrix whose rows form the basis for the dual space of the codewords. In other words, the rows of H are orthogonal to the codewords. Consequently, if x is the trnasmitted codeword and z is the additive noise vector caused by the channel, the receiver's input would be $y = x + z$.[1] Now the receiver could compute the *syndrome* $s = Hy = Hx + Hz = Hz$, since $Hx = 0$. As a result, what is left is purely the effect of noise.

The next step would obviously be to find an efficient method to obtain z from $s = Hz$. There are numerous decoding algorithms designed for this purpose. However, one particular family is of special interest for us and algorithms related to neural networks: graph-based codes.

Graph-based Codes

Graph-based codes are a particular family of linear codes where the parity check matrix represents the connectivity of a bipartite graph. This graph plays a key role in the decoding process, as shall be seen shortly. Some coding algorithms among this family, such as Low Density Parity Check (LDPC) [17] and Raptor [18] codes, are very efficient and have been shown to approach channel capacity for various settings.

[1]Recall that all operations are performed in the binary field $GF(2)$.

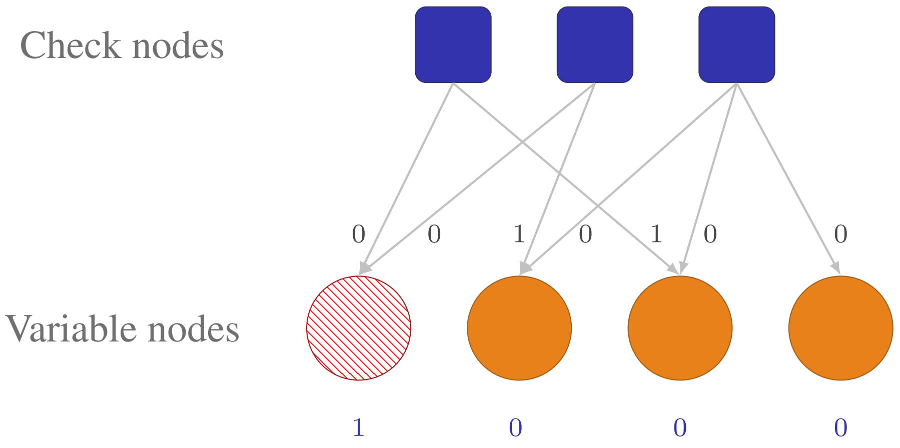


Figure 3.2: Gallager's decoding algorithm for LDPC codes. Here, $n = 4$ and $k = 3$.

However, what makes graph-based codes interesting for us is the way they achieve this degree of efficiency. The retrieval (decoding) process in graph-based codes involves a series of message-passing over the often-sparse parity check graph. As such, this process has similarities to the way neurons exchange messages over the neural graph in order to accomplish their tasks.

To better understand the decoding mechanism for graph-based codes, let us consider a simple algorithm design by Gallager for LDPC codes [19]. To this end, consider a bipartite graph as shown in Figure 3.2. In the graph, the lower and upper nodes are called *variable* and *check* nodes, respectively. The graph has n variable and $n - k$ check nodes, which correspond to the bits of the codeword and syndrome, respectively.

The decoding algorithm starts by initializing the variable nodes with the received input from the channel. The variable nodes send these values to their neighboring check nodes. Then, the algorithm proceeds in rounds in each of which the variable node v_i and its neighbor, check node c_j, exchange the following set of messages:

1. c_j send the modulu-2 sum of the received bits among its *other* neighbors to v_i. This shows the *belief* of c_j about the correct state of v_i.

2. v_i sends its initial state (received from the channel) unless *all* (or in another version, the majority of) its neighbors tell him to change its state. In that case, v_i send the new state.

The algorithm continues until the variable nodes converge to a state orthogonal to the parity check matrix or a decoding failure occurs.

In the example shown in Figure 3.2, suppose the transmitted codeword was $x = (0, 0, 0, 0)$ and the first bit (the red hatched node) is changed to 1 due to channel noise. The messages that are sent by the check nodes to each variable node are shown beside each edge. It is easy to see that in the next round, the first variable node changes its state to 0 as all its neighbors unanimously agree on the correct state and the algorithm finishes successfully. This simple example can in fact be generalized to much more sophisticated situations with the success of the algorithm is still guaranteed. They show the power of simple iterative methods over a properly designed graph.

The design process for an efficient error correcting code usually boils down to two crucial steps:

1. The choice of the graph, i.e., its degree distribution,

2. The decoding algorithm.

Both criteria have profound effects on the performance of the overall method.

Differences with Neural Algorithms

In comparison to the above two criteria, designing a neural message passing algorithm for associative memories is different in some key aspects. First and most important of all, the retrieval (decoding) algorithm has to be simple enough to comply with limitations of neurons. These limitations are

1. In constraint to variable or check nodes, a neuron can not transmit different messages over different edges. What a neuron transmits goes to all its neighbors identically.

2. In our model, neurons could not have access to individual messages received over their input links. The only decision parameter that they have is the weighted input sum plus some internal threshold.

3. Neurons do not operate in the binary $GF(2)$ field.

4. In contrast to many state-of-the-art decoding algorithm, such as belief propagation, the messages that neurons transmit have limited precision. In many cases, the messages are binary or the number of spikes in a short time interval.

5. In coding techniques, the codewords could be designed at will, whereas in neural networks the codewords (patterns) are given and should be learned.

The second difference in the design process comes from the fact that explicit construction of neural graphs (or their ensemble) is usually not an option as the graph is *learned* from examples that we might not have any control over. As a result, one is usually faced with analyzing the behavior of the algorithm given certain general properties of the graph, e.g., the degree of its sparsity.

In the rest of this work, our main objective is to design efficient algorithms that perform the learning and recall phases of a neural associative memory when the set of input patterns forms a subspace. The algorithms comply with the aforementioned neural limitations. We then borrow some theoretical techniques and utilize them to analyze the performance of the proposed algorithms. And finally, some simulations are conducted to verify the sharpness of theoretical analysis in practice.

Part II

Early Attempts

Chapter 4

Memorizing Low Correlation Sequences

To improve the pattern retrieval capacity of associative memories, we start by taking a closer look at one of the cornerstones of the filed: the Hopfield network [21], and its recall phase. On a closer inspection, we notice that the recall phase reduces to calculating the "correlation" between the given cue and the memorized patterns. In doing so, the input sum that each neuron receives can be divided into two parts: the desired term and the undesired interference caused by the correlation between the memorized patterns and the cue.

That is a good starting point: instead of memorizing any set of random patterns, let us focus on patterns that have some minimal correlation. This way, we might be able to increase the pattern retrieval capacity of associative memories similar to the Hopfield network. However, as we will see by the end of this chapter, although the considered family of low-correlation sequences improve the pattern retrieval capacity, it is still far away from exponential efficiencies we are looking for in this thesis.

4.1 Related Work

Designing a neural associative memory has been an active area of research for the past three decades. Hopfield was among the first to design an artificial neural associative memory in his seminal work in 1982 [21]. The so-called Hopfield network is inspired by Hebbian

The content of this chapter is joint work with Raj K. Kumar, Amin Shokrollahi and Wulfram Gerstner. It was published in [20].

learning [22] and is composed of binary-valued (± 1) neurons, which together are able to memorize a certain number of patterns. The learning rule for a Hopfield network is given by

$$w_{ij} = \frac{1}{n} \sum_{\mu=1}^{C} x_i^{\mu} x_j^{\mu}, \tag{4.1}$$

where w_{ij} is the weight between neurons i and j in the neural graph, x_i^{μ} is the i^{th} bit of pattern μ, n is the length of the patterns and C is the total number of patterns.

The pattern retrieval capacity of a Hopfield network of n neurons was derived later by Amit et al. [23] and shown to be $0.13n$, under vanishing bit error probability requirement. Later, McEliece et al. [24] proved that under the requirement of vanishing pattern error probability, the capacity of Hopfield networks is $n/(2\log(n))) = O(n/\log(n))$.

In addition to neural networks with online learning capability, offline methods have also been used to design neural associative memories. For instance, in [25] the authors assume that the complete set of patterns is given in advance and calculate the weight matrix using the pseudo-inverse rule [13] offline. In return, this approach helps them improve the capacity of a Hopfield network to $n/2$, under vanishing pattern error probability condition, while being able to correct *one bit* of error in the recall phase. Although this is a significant improvement over the $n/\log(n)$ scaling of the pattern retrieval capacity in [24], it comes at the price of much higher computational complexity and the lack of gradual learning ability.

While the connectivity graph of a Hopfield network is a complete graph, Komlos and Paturi [26] extended the work of McEliece to sparse neural graphs. Their results are of particular interest as physiological data is also in favor of sparsely interconnected neural networks. They have considered a network in which each neuron is connected to d other neurons, i.e., a d-regular network. Assuming that the network graph satisfies certain connectivity measures, they prove that it is possible to store a linear number of *random* patterns (in terms of d) with vanishing bit error probability or $C = O(d/\log n)$ random patterns with vanishing pattern error probability. Furthermore, they show that in spite of the capacity reduction, the error correction capability remains the same as the network can still tolerate a number of errors which is linear in n.

It is also known that the capacity of neural associative memories could be enhanced if the patterns are of *low-activity* nature, in the sense that at any time instance many of the neurons are silent [6]. More specifically, if $p < 0.5$ is the probability that a neuron fires, then it can be shown that the pattern retrieval capacity scales with $n/(p\log p)$. As p goes towards zero, the pattern retrieval capacity increases. However, it is still linear in n [27,28].[1] Furthermore, these schemes fail when required to correct a fair amount of erroneous bits as the information retrieval is not better compared to that of normal networks [13].

[1]Additionally, the total number of patterns that have roughly pn neurons decreases when $p \to 0$.

Given that even very complex offline learning methods can not improve the capacity of binary or multi-state neural associative memories, a group of recent works has made considerable efforts to exploit the inherent structure of the patterns in order to increase capacity and improve error correction capabilities. Such methods focus merely on memorizing those patterns that have some sort of inherent redundancy. As a result, they differ from previous methods in which the network was designed to be able to memorize any random set of patterns. Pioneering this approach, Berrou and Gripon [29] achieved considerable improvements in the pattern retrieval capacity of Hopfield networks, by utilizing Walsh-Hadamard sequences. Walsh-Hadamard sequences are a particular type of low correlation sequences and were initially used in CDMA communications to overcome the effect of noise. The only slight downside to the proposed method is the use of a decoder based on the winner-take-all approach which requires a separate neural stage, increasing the complexity of the overall method.

In this chapter, we propose a neural association mechanism that employs binary neurons to memorize patterns belonging to another type of low correlation sequences, called Gold family [30]. The network itself is very similar to that of Hopfield, with a slightly modified weighting rule. Therefore, similar to a Hopfield network, the complexity of the learning phase is small. However, we can not increase the pattern retrieval capacity beyond n and show that the pattern retrieval capacity of the proposed model is $C = n$, while being able to correct a fair number of erroneous input bits.

4.2 Problem Formulation and the Model

The neural network model that we consider can be represented by an undirected complete graph of binary nodes with weighted links between each pair of nodes. Each node of the graph represents a neuron. At every instance of the process the state of node i, denoted by s_i, indicates whether or not neuron i has fired at that instance (we will not show the dependency of s_i on time to alleviate notation). In particular, we set $s_i = 1$ when the neuron fires, and $s_i = -1$ when it remains silent. The weight w_{ij} between nodes i and j denotes the binding strength (interaction) between these two neurons and can assume any real number.

At any given instance, a node in the network decides its state based on the inputs from its neighbors. More specifically, neuron i fires if the weighted sum

$$h_i = \sum_{j=1}^{n} w_{ij} s_j \tag{4.2}$$

over its input links w_{ij} exceeds a firing threshold Θ,

$$s_i = \begin{cases} 1, & \text{if } h_i \geq \Theta \\ -1, & \text{otherwise} \end{cases} . \tag{4.3}$$

The main task of neural association is to choose the graph weights w_{ij} such that the network is able to memorize C binary patterns of length n. Here, we are mostly interested in auto-association, i.e., retrieving a memorized pattern from its noisy version. In the sequel, we denote these patterns by $\{x^1, x^2, \ldots, x^C\}$, where $x^\mu = (x_1^\mu \cdots x_n^\mu)$ is a binary pattern of length n with $x_j^\mu \in \{-1, +1\}$ for all j.

4.3 The Role of Correlation in the Hopfield Network

Consider the case where we use the Hopfield weighting rule (4.1). To gain better insight on the role of correlation in the recall phase of a Hopfield network, consider the input sum received by neuron i (h_i in equation (4.2)), when we initialize the network with pattern x^γ:

$$\begin{aligned} h_i &= \sum_{j=1}^{n} w_{ij} x_j^\gamma = \frac{1}{n} \sum_{\mu=1}^{C} x_i^\mu \sum_{j=1}^{n} x_j^\mu x_j^\gamma \\ &= x_i^\gamma + \frac{1}{n} \sum_{\substack{\mu=1 \\ \mu \neq \gamma}}^{C} x_i^\mu \langle x^\mu, x^\gamma \rangle =: x_i^\gamma + I_i^\gamma, \end{aligned} \tag{4.4}$$

where $\langle x^\mu, x^\gamma \rangle$ is the inner product of patterns x^γ and x^μ. In the above equation, the first term, i.e., x_i^γ, is the "desired term" because we would like the thresholded version of h_i (according to (4.3)) to be equal to the x_i^γ (since our objective is to recall pattern x^γ). The second term I_i^γ is the interference term, which depends on the correlation between pattern γ and all other patterns. Ideally, we would like the interference term be as small as possible such that it allows for recovery of x^γ.

Therefore, in order to improve the pattern retrieval capacity, one idea is to focus on patterns that have low correlation among each other, so that the undesired interference term is minimized.

4.4 Low Correlation Sequences

Summations similar to (4.4) also appear in CDMA communications and there is a rich theoretical background on the applications of low correlation patterns in reducing interference in CDMA networks. One such family of sequences[2] is the set of Walsh-Hadamard sequences. Recently, this family was employed by Berrou and Gripon [29] to improve the pattern retrieval capacity of Hopfield networks. However, this approach involves a separate soft Maximum Likelihood (ML) decoder to deal with noise in the input, which increases the complexity of the network significantly.

[2]In this work, we use the terms pattern and sequence to convey the same meaning.

Furthermore, it can be shown that the traditional CDMA approach of employing any family of low-correlation sequences achieving the Welch bound is insufficient for our purpose (see Appendix 4.A for more details); smarter cancellation is needed among the summands of the second term in (4.4).

In what follows, we will show that such intelligent cancellation occurs when we suitably pick the x^{μ} from the family of Gold sequences [30]. We continue with a quick overview of this sequence family.

Gold Sequences

In this subsection, we assume that the reader is familiar with the basics of finite fields. Let $q > 0$ be an odd integer, and $d = 2^l + 1$ for some l such that l and q are relatively prime. Also, let α be a primitive element of $\mathbb{F}_{2^q}$, and $T(\cdot)$ denote the finite field trace function from $\mathbb{F}_{2^q}$ to $\mathbb{F}_2$. We set $n = 2^q - 1$. The family of cyclically distinct Gold sequences is defined to be the set $\mathcal{G} = \{g^{(a)} | a \in \mathbb{F}_{2^q}\}$, where each $g^{(a)} = (g_1^{(a)} \cdots g_n^{(a)})$ is a sequence of length n with

$$g_i^{(a)} \triangleq (-1)^{T(a\alpha^i)+T(\alpha^{di})}. \tag{4.5}$$

Hence the number of sequences in $\mathcal{G}$ is $n+1$. For later use, we define the notation $g^{(a)}(k)$ to represent a cyclic shift of $g^{(a)}$ by k positions (for simplicity, we will refer to $g^{(a)}(0)$ simply as $g^{(a)}$). Also, it can be easily verified that the Gold sequences are periodic with period n [30]. In the sequel, we will choose our patterns $\{x^1, \ldots, x^C\}$ to be either Gold sequences, or cyclic shifts of Gold sequences, depending on the value of C. The particular sequences chosen will be specified later. We set the first pattern $x^1 = g^{(0)}$, where 0 denotes the zero element in $\mathbb{F}_{2^q}$.

Before we proceed, it must be noted that generating low correlation sequences using neural networks is easily accomplished, since one can generate the Gold family using only shift registers and logical AND operations [30] (the logical AND operation may be implemented using a simple two-layer neural network, see [13]). Owing to lack of space, we will not go into much details in this regard and point the reader to [30], [31] for further details regarding how Gold sequences are generated using shift registers.

4.5 Gold Sequence and Hopfield Networks with Scaled Weights

Now that the patterns are chosen from the Gold family, we show how a variant of Hopfield network could achieve a higher capacity. The main idea in the proposed method is to consider a generalized learning rule,

$$w_{ij} = \frac{1}{n} \sum_{\mu=1}^{C} \lambda_\mu x_i^\mu x_j^\mu, \tag{4.6}$$

where the scaling factors λ_μ are to be picked such that the interference term in (4.4) becomes small. To demonstrate the necessity of λ_μ's, we first examine the case of $\lambda_\mu = 1, \ \forall \mu = 1, \ldots, C$, which reduces (4.6) to the original Hopfield weighting rule in (4.1). We will show that this scheme has very good performance in terms of stability but that the error correction capability is very poor. As a result, we propose a way of choosing λ_μ's that ensures fair noise tolerance as well as stability of the network.

Stability Analysis with $\lambda_\mu = 1 \ \forall \ \mu$

To assess the stability property of Gold sequences, we pick the $C = n + 1$ sequences from $\mathcal{G}$ to constitute the patterns that we would like to memorize (these are all cyclically distinct, by construction). Without loss of generality, let us assume that the pattern x^γ, which we would like to recall at the moment, corresponds to the Gold pattern with $a = 0$ (see (4.5)). Specializing the second term of (4.4) for the case of our patterns being Gold sequences, we obtain

$$\begin{aligned}
I_i^\gamma &= \frac{1}{n} \sum_{a \in \mathbb{F}_{2^q}, \ a \neq 0} (-1)^{T(a\alpha^i) + T(\alpha^{di})} \sum_{j=1}^{n} (-1)^{T(a\alpha^j)} \\
&= \frac{(-1)^{T(\alpha^{di})}}{n} \left(C - 1 + \sum_{j=1, \neq i}^{n} \sum_{\substack{a \in \mathbb{F}_{2^q} \\ a \neq 0}} (-1)^{T(a(\alpha^i + \alpha^j))} \right) \\
&= \frac{C-n}{n} (-1)^{T(\alpha^{id})} = \frac{C-n}{n} x_i^\gamma = \frac{1}{n} x_i^\gamma.
\end{aligned} \tag{4.7}$$

The first equality in (4.7) holds because $\alpha^i + \alpha^j \neq 0$ for $i \neq j$ (since α is primitive element of $F(2^q)$), and then since the trace function is a linear form:

$$\sum_{a \in \mathbb{F}_{2^q}} (-1)^{T(ab)} = 0, \text{ if } b \neq 0. \tag{4.8}$$

Combining (4.7) and (4.4), we see that for the Gold family with $C = n + 1$ we have $h_i = \frac{C}{n} x_i^\gamma$. Therefore, the sign of h_i is equal to that of x_i^γ, i.e., the network is stable for $C = n + 1$ when Gold sequences are memorized. Furthermore, we can also show that setting C to be an integer multiple of $(n + 1)$, say $C = \delta(n + 1)$, $\delta \in \mathbb{Z}^+$, and picking our patterns to be $\mathcal{G}$ and $(\delta - 1)$ sets of cyclic shifts of all sequences in $\mathcal{G}$ will also lead to stability.

Performance in the Presence of Noise when $\lambda_\mu = 1 \; \forall \; \mu$

To investigate the performance in the presence of noise, we first consider the simple case of a single bit error in the input to the network. Suppose that we feed in the pattern x^γ which corresponds to the Gold sequence with $a = 0$ (see (4.5)), where position k has been flipped. In other words, our initial pattern is $s = x^\gamma + e$, where the error vector e is equal to zero in all positions except for the k^{th} element e_k, and $e_k = -2x_k^\gamma$. Substituting this input pattern into (4.4) and using (4.7), we obtain:

$$h_i = \frac{C}{n} x_i^\gamma - \frac{2}{n} \sum_{\mu=1}^{C} x_i^\mu x_k^\mu x_k^\gamma. \tag{4.9}$$

The term $E_k^\gamma := \sum_{\mu=1}^{C} x_i^\mu x_k^\mu x_k^\gamma$ reduces to Cx_i^γ if $k = i$. When $k \neq i$, we obtain using (4.8) that $E_k^\gamma = 0$. From (4.9) and the above, we obtain

$$h_i = \begin{cases} -\frac{C}{n} x_i^\gamma, & \text{if } i = k \\ \frac{C}{n} x_i^\gamma, & \text{Otherwise} \end{cases}. \tag{4.10}$$

This essentially means that no error correction is possible: all flipped bits remain erroneous and all correct bits remain correct as the system evolves.

Choosing λ_μ to Ensure Good Error Correction

From the above, the need to choose the λ_μ intelligently becomes clear: we need to make sure that pair-wise pattern correlations cancel each other out when summing up over all patterns, in the second term of (4.4). Towards this end, we simply pick $\lambda_\mu = (-1)^{T(\alpha^{-\mu})}$; since this term alternates in sign for different μ's, the interference terms in the summation in (4.4) cancel each other out. This intuition is validated by the simulation results below, where we show that using this choice of λ_μ, we will be able to memorize $C = n$ patterns drawn from the Gold sequence family.

4.6 Simulation Results

We consider three different Gold families, with parameters $q = 5$, $q = 7$ and $q = 9$. These correspond to having $n = 31$, $n = 127$ and $n = 511$ respectively. As mentioned earlier, the first $C = n$ patterns of the Gold family are selected. We consider various numbers of initial (uniformly random) bit flips and evaluate the pattern error rate. The results are reported in Figure 4.1 and are compared to the pseudo-inverse method proposed in [25].[3] As can be seen,

[3]For clarity purposes, we have illustrated only the pattern error rate corresponding to $n = 511$ for the pseudo-inverse method [25] as the results for $n = 31$ and $n = 127$ were quite the same.

the number of initial bit flips that can be corrected by the proposed method depends on the network size n. With larger n's, we are able to tolerate larger amounts of noise. For instance, with $n = 31$, the suggested method is able to correct one bit flip, while with $n = 511$, the proposed solution can correct up to 40 bits of error. Hence, our method results in a robust associative memory with $C = n$, which is significantly better than previous works both in terms of capacity [24] and noise tolerance [25].

4.7 Final Remarks

In this chapter, we showed how focusing on a particular family of low correlation sequences will improve the pattern retrieval capacity. In [20], we also proposed another approach based on Gold sequences over bidirectional associative memories. The proposed model was better in terms of error correction as it correct up to $\lfloor (n - \sqrt{2(n+1)} + 1)/2 \rfloor$ initial errors. However, to memorize C patterns of length n we need a network of size $C + n$. However, the proposed approaches are still incapable of achieving exponential pattern retrieval capacities we are interested in.

We also made several attempts to find an analytical lower bound on the number of errors that the proposed scaled Hopfiled network can correct when $\lambda_\mu = (-1)^{T(\alpha^{-\mu})}$. Unfortunately, the analysis was not straightforward. Ideas based on finding the roots of elliptic curves seemed promising. However, by that time we had our eyes on a new model based on memorizing patterns that belong to a subspace, as will be discussed in the following chapters.

4.A Borrowing ideas from CDMA systems

From a cursory examination of (4.4), the similarities of the Hopfield network stability condition with system design equations for code division multiple access (CDMA) systems [32] is evident (in the context of CDMA systems, C would correspond to the number of users, and L to the pseudorandom spreading sequence length). This similarity was also noticed in [29], as mentioned previously. The traditional approach adopted by the CDMA community to solve the neural stability problem would be to choose the set of patterns $\{x^\mu\}_{\mu=1}^{C}$ from a family of low correlation sequences [31], such that the magnitude of the correlation $|\langle x^\mu, x^\gamma \rangle|$ is small for all μ, γ with $\mu \neq \gamma$. However, as we shall argue in the sequel, our requirements turn out to be a bit more stringent than those encountered in traditional CDMA system design.

Let us first focus on the case of $C \gg n$. In order to lower bound the magnitudes of the correlations $|\langle x^\mu, x^\gamma \rangle|$, we recall the Welch bound on cross-correlation [33].

Theorem 1 (Welch). *Let $\{(x_i^\mu)_{i=1}^{n}\}$, $\mu = 1, 2, \ldots, C$, be a collection of C vectors of length*

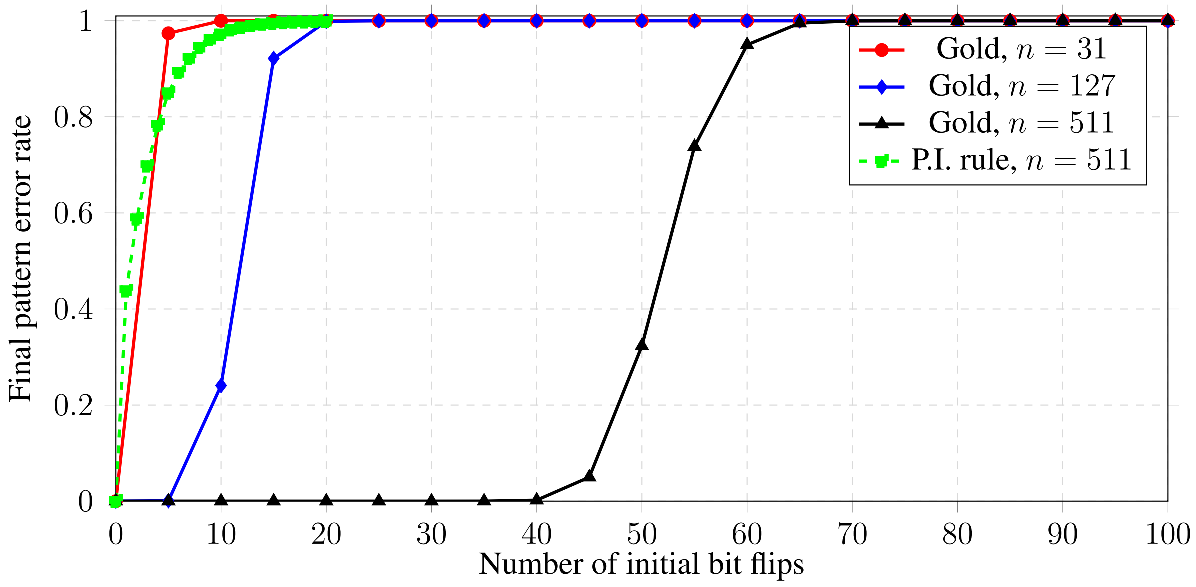


Figure 4.1: Pattern error rate for our scaled Hopfield network with $C = n$, compared to the pseudo-inverse method [25] with $C = \lfloor n/2 \rfloor - 1$.

n, satisfying

$$\forall \, \mu : \sum_{i=1}^{n} |x_i^\mu|^2 = n$$

and let

$$\theta_{\max} := \max \left| \sum_{i=1}^{n} x_i^\mu (x_i^\nu)^* \right|, \; \mu \neq \nu,$$

where $(\cdot)^*$ denotes the complex conjugate operation. Then, for $k = 1, 2, \ldots$

$$\theta_{\max} \geq \frac{n^{2k}}{C-1} \left[\frac{C}{\binom{n-1+k}{k}} \right].$$

For the special case of $k = 1$ we have

$$\theta_{\max} \geq \sqrt{n} \sqrt{\frac{C-n}{C-1}}. \tag{4.11}$$

Traditional CDMA system design would suggest that we choose our patterns x^μ from a family of sequences that achieves the Welch bound (i.e., we have equality in (4.11)). Supposing that we do this, the magnitude of the interference term $|I_i^\gamma|$ from (4.7) evaluates to

$$\begin{aligned}|I_i^\gamma| &= \frac{1}{n}\left|\sum_{\mu=1,\neq\gamma}^{C} x_i^\mu \langle x^\mu, x^\gamma\rangle\right| \\ &\leq \frac{1}{n}(C-1)\sqrt{n}\sqrt{\frac{C-n}{C-1}} \\ &= \sqrt{\frac{(C-n)(C-1)}{n}}\end{aligned}$$

From (4.4), we see that a sufficient condition for pattern x^γ to be stable is that $\sqrt{\frac{(C-n)(C-1)}{n}} < 1$, which is equivalent to $C < n+1$. This is contrary to our initial assumption of $C \gg n$. Note that while the above analysis is not rigorous (since it involves analyzing an upper bound on $|I_i^\gamma|$ that might not be tight), it does give us important insight on how we may design the patterns $\{x^\mu\}$ intelligently: we not only need to ensure that the correlation terms in I_i^γ are low, we also need to design the x^μ such that summands in I_i^γ cancel out each other well.

Part III

Linear Regularities to the Rescue

Chapter 5

Subspace Learning and Non-Binary Associative Memories

Although low correlation sequences were giving us an increase in the pattern retrieval capacity, they were still insufficient for bridging the gap between linear to exponential capacities. As a result, we decided to change the model. Given that linear codes achieve the exponential capacities by forming a subspace in the n-dimensional space, we considered non-binary patterns that come from a subspace in our new model. We started by considering a bidirectional associative memory and showed that if the neural graph is an expander (formally defined in Appendix 5.B) and is given, then we could achieve exponential pattern retrieval capacities while being able to correct two initial erroneous symbols in the recall phase. We later extended this result to any sparse neural graph and proposed a learning algorithm to provide us with the suitable neural graph.

For brevity, we jump directly to the latter result and explain the steps in more details (technical details regarding the earlier models based on the expander graphs are given in Appendix 5.B for the sake of completeness). In particular, we propose a learning algorithm for the case when patterns belong to a subspace. We show that if the learned connectivity matrix W has certain properties, we could correct a constant number of errors in the recall phase. Furthermore, we prove that the proposed approach is capable of memorizing an exponential number of patterns if they belong to a subspace.

The content of this chapter is joint work with Raj K. Kumar and Amin Shokrollahi. It was published in [34] and [35].

5.1 Problem Formulation and the Model

The problem is the same as what we had in the previous chapter: find the connectivity matrix W of the neural graph such that the patterns in a given database $\mathcal{X}_{n\times C}$ are stable states of the network. Furthermore, the resulting neural associative memory should be fairly noise resilient.

However, the model which we consider in this chapter differs from that of the previous chapters in two key aspects:

1. Non-binary neurons: Here, we work with neurons whose states are integers from a finite set of non-negative values $\mathcal{Q} = \{0, 1, \ldots, Q-1\}$. A natural way of interpreting this model is to think of the integer states as the short-term firing rate of neurons (possibly quantized). In other words, the state of a neuron in this model indicates the number of spikes fired by the neuron in a fixed short time interval.

2. Patterns form a subspace of $\mathbb{R}^n$: In other words, if $\mathcal{X}_{C\times n}$ is the matrix that contains the set of n-dimensional patterns in its rows, with entries in $\mathcal{Q}$, then we assume rank of this matrix to be $k < n$.

These two properties help us design neural associative memories that achieve exponential pattern retrieval capacities. The neural weights are assumed to be real-valued, as was the case in the previous chapter.

Note that if $k = n$ and $\mathcal{Q} = \{-1, 1\}$, then we are back to the original associative memory problem [21]. However, our focus will be on the case where $k < n$, which will be shown to yield much larger pattern retrieval capacities.

Solution Overview

Before going through the details of the algorithms, let us give an overview of the proposed solution.

The learning phase

Since the patterns are assumed to belong to a subspace, in the learning phase memorize the dataset $\mathcal{X}$ by finding a set of non-zero vectors $w_1, \ldots, w_m \in \mathbb{R}^n$, with $m \leq n-k$, that are orthogonal to the set of given patterns. Note that such vectors are guaranteed to exist, one example being a basis for the null-space. To learn the set of given patterns, we have adopted the neural learning algorithm proposed in [36] and modified it to favor sparse solutions (we will see shortly why sparseness of the neural graph is necessary). In each iteration of the algorithm, a random pattern from the data set is picked and the neural weights corresponding

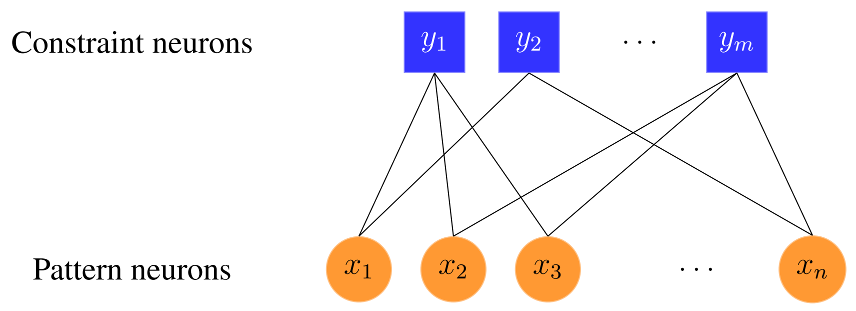


Figure 5.1: A bipartite graph that represents the constraints on the training set. The weights are bidrectional. However, depending on the recall algorithm (which is explained later), the weight from y_i to x_j (W^b_{ij}) could be either equal to the weight from x_j to y_i (W_{ij}) or its sign. In other words, we have either $W^b_{ij} = \text{sign}(W_{ij})$ or $W^b_{ij} = W_{ij}$, depending on the algorithm used in the recall phase.

to constraint neurons are adjusted is such a way that the projection of the pattern along the current weight vectors is reduced, while trying to make the weights sparse as well.

As a result, the inherent structure of the patterns is captured in the obtained null-space vectors, denoted by the matrix $W \in \mathbb{R}^{m\times n}$, whose i^{th} row is w_i. This matrix can be interpreted as the adjacency matrix of a bipartite graph which represents our neural network. The graph is comprised of pattern and constraint neurons (nodes). Pattern neurons, as their name suggests, correspond to the states of the patterns we would like to learn or recall. The constrain neurons, on the other hand, should verify if the current pattern belongs to the database $\mathcal{X}$. If not, they should send proper feedback messages to the pattern neurons in order to help them converge to the correct pattern in the dataset. The overall network model is shown in Figure 5.1.

The recall phase

In the recall phase, the neural network retrieves the correct memorized pattern from a possibly corrupted version. In this case, the states of the pattern neurons are initialized with a corrupted version of pattern, say, x^μ, i.e. $x = x^\mu + z$, where z is the noise.

To eliminate the noise vector z, the algorithm relies on two properties of the neural graph during the recall process:

1. The neural weights are orthogonal to the patterns (i.e $Wx^\mu = 0$). As a result, the net input to the constraint neurons will be caused only by noise (i.e. $Wx = Wz$).

2. The graph is sparse, which makes it possible to identify the corrupted pattern neurons from the rest and correct them.

The second property helps us find which pattern neurons are corrupted and the first one helps us estimate the value of noise at these particular places.

The proposed recall algorithm is iterative and in each iteration, the constraint neurons send some feedback to their neighboring pattern neurons, to tell them if they need to update their state and in which direction. The pattern neurons that receive some feedback from the majority of their neighbors update their state accordingly in the hope of eliminating input noise. This process continues until noise is eliminated completely or a failure is declared.

Pattern retrieval capacity

Finally, we are going to show that the proposed model is capable of memorizing an exponentially large number of patterns. The idea behind this proof comes from the fact that there are sets $\mathcal{X}$ with exponentially many integer-valued patterns that form a subspace.

5.2 Related Work

In the previous chapter, we discussed previous work on binary neural associative memories. Extension of associative memories to non-binary neural models has also been explored in the past. Hopfield addressed the case of continuous neurons and showed that similar to the binary case, neurons with states between -1 and 1 can memorize a set of random patterns, albeit with less capacity [37]. In [38] the authors investigated a multi-state complex-valued neural associative memory for which the estimated capacity is $C < 0.15n$. Under the same model but using a different learning method, Muezzinoglu et al. [39] showed that the capacity can be increased to $C = n$. However the complexity of the weight computation mechanism is prohibitive. To overcome this drawback, a Modified Gradient Descent learning Rule (MGDR) was devised in [40].

Using patterns from a low rank subspace to increase the pattern retrieval capacity has also been considered by Gripon and Berrou [41]. They have come up with an approach based on neural cliques, which increases the pattern retrieval capacity to $O(n^2)$. Their method is based on dividing a neural network of size n into c clusters of size n/c each. Then, the messages are chosen such that only one neuron in each cluster is active for a given message. Therefore, one can think of messages as a random vector of length $c\log(n/c)$, where the $\log(n/c)$ part specifies the index of the active neuron in a given cluster. The authors also provide a learning algorithm, similar to that of Hopfield [21], to learn the pairwise correlations within the patterns. Using this technique and exploiting the fact that the resulting patterns are very sparse, they could boost the capacity to $O(n^2)$ while maintaining the computational simplicity of Hopfield networks.

In contrast to the pairwise correlation of the Hopfield model, Peretto et al. [42] deployed *higher order* neural models: the models in which the state of the neurons not only depends on the state of their neighbors, but also on the correlation among them. Under this model, they showed that the storage capacity of a higher-order Hopfield network can be improved to $C = O(n^{p-2})$, where p is the degree of correlation considered. The main drawback of this model is the huge computational complexity required in the learning phase, as one has to keep track of $O(n^{p-2})$ neural links and their weights during the learning period.

The proposed model in this chapter can be also thought of as a way to capture higher order correlations in given patterns while keeping the computational complexity to a minimal level (since instead of $O(n^{p-2})$ weights one needs to only keep track of $O(n^2)$ of them). Furthermore, the pattern retrieval capacity of the proposed model is exponential, although it does not have the generality of the method proposed in [42] as it can only memorize patterns that belong to a subspace.

An important point to note is that learning linear constraints by a neural network is hardly a new topic as one can learn a matrix orthogonal to a set of patterns in the training set (i.e., $Wx^\mu = 0$) using simple neural learning rules (we refer the interested readers to [43] and [44]). However, to the best of our knowledge, finding such a matrix subject to the sparsity constraints has not been investigated before. This problem can also be regarded as an instance of compressed sensing [45], in which the measurement matrix is given by the big patterns matrix $\mathcal{X}_{C\times n}$ and the set of measurements are the constraints we look to satisfy, denoted by the tall vector b, which for simplicity reasons we assume to be all zero. Thus, we are interested in finding a sparse vector w such that $\mathcal{X}w = 0$. Nevertheless, many decoders proposed in this area are very complicated and cannot be implemented by a neural network using simple neuron operations. Some exceptions are [46] and [47] which are closely related to the learning algorithm proposed in this paper.

5.3 Learning Phase

Since the patterns are assumed to be coming from a subspace in the n-dimensional space, we adapt the algorithm proposed by Oja and Karhunen [36] to learn a basis of the dual space of the subspace defined by the patterns. In fact, a very similar algorithm is also used in [43] for the same purpose. However, since we need the basis vectors to be sparse (due to requirements of the algorithm used in the recall phase), we add an additional term to penalize non-sparse solutions during the learning phase.

Another difference between the proposed method and that of [43] is that the learning algorithm proposed in [43] yields dual vectors that form an orthogonal set. Although one can easily extend our suggested method to such a case as well, we find this requirement unnecessary in our case. This gives us the additional advantage to make the algorithm *parallel* and *adaptive*: parallel in the sense that we can design an algorithm to learn one constraint and repeat it several times in order to find all constraints with high probability; and adaptive in the sense that we can determine the number of constraints on-the-go, i.e., start by learning just a few constraints. If needed (for instance due to bad performance in the recall phase), the network can easily learn additional constraints. This increases the flexibility of the algorithm and provides a nice trade-off between the time spent on learning and the performance in the recall phase. Both these points make an approach biologically realistic.

The proposed algorithm in this chapter is online, i.e., the algorithm learns gradually and iteratively from examples in the dataset. We also proposed an offline learning algorithm in [48] which we do not discuss here for brevity.

Overview of the proposed algorithm

The problem to find one sparse constraint *vector* w is given by equations (5.1a), (5.1b), in which pattern μ is denoted by x^μ.

$$\min \sum_{\mu=1}^{C} |w^\top x^\mu|^2 + \eta g(w). \tag{5.1a}$$

subject to:

$$\|w\|_2 = 1 \tag{5.1b}$$

In the above problem, $\|.\|_2$ represent the ℓ_2 vector norm, $g(w)$ a penalty function to encourage sparsity and η is a positive constant. There are various ways to choose $g(w)$. For instance one can pick $g(w)$ to be $\|.\|_1$, which leads to ℓ_1-norm penalty and is widely used in compressed sensing applications [46], [47]. Here, we will use a different penalty function, as explained later.

To form the basis for the null space of the patterns, we need $m = n - k$ vectors, which we can obtain by solving the above problem several times, each time from a random initial point[1].

Remark 2. *Note that in order to have a sparse graph, the pattern and constraint neurons should have degrees that are $O(1)$. This implies that $m = O(n)$. However, when the goal is to correct only a constant number of errors, we could have $m = O(1)$ under special circumstances. Nevertheless, in this thesis we focus only on the cases where $m = O(n)$.*

[1]It must be mentioned that in order to have exactly $m = n - k$ linearly independent vectors, we should pay some additional attention when repeating the proposed method several times. This issue is addressed later in the chapter.

As for the sparsity penalty term $g(w)$ in this problem, we consider the function

$$g(w) = \sum_{i=1}^{n} \tanh(\sigma w_i^2),$$

where σ is chosen appropriately. Intuitively, $\tanh(\sigma w_i^2)$ approximates $|\text{sign}(w_i)|$ and, therefore, $g(w)$ will approximate ℓ_0-norm. Trivially, the larger σ is, the closer $g(w)$ will be to $\|.\|_0$.

To solve problem (5.1), one can either use standard (batch) gradient descent or stochastic gradient descent. In standard gradient descent, we iteratively update w towards minimizing (5.1) and in each iteration, the amount of update is proportional to the derivative of (5.1a) with respect to w. The stochastic gradient descent is virtually the same, except for the fact that the amount of update in each iteration is proportional to the derivative of $|x^\mu \cdot w|^2 + \eta g(w)$ for a randomly picked pattern x^μ.

Each of these approaches has their pros and cons. For instance, the standard approach is usually faster and converges more easily. However, the stochastic approach is more suitable for neural learning algorithms as it requires to "see" one sample at a time. This makes gradual learning possible as we do not need to have the whole dataset $\mathcal{X}$ available first before being able to calculate the updates in the standard gradient descent method. Therefore, we will focus on the stochastic version here. A similar learning algorithm based on standard (offline) gradient descent is discussed in [48].

Assuming $x(t)$ to be the randomly picked pattern in iteration t of the stochastic gradient descent approach, we could calculate the derivative of the objective function with respect to w to get the following iterative algorithm:

$$y(t) = x(t) \cdot w(t) \tag{5.2a}$$

$$\tilde{w}(t+1) = w(t) - \alpha_t \left(2y(t)x(t) + \eta\Gamma(w(t))\right) \tag{5.2b}$$

$$w(t+1) = \frac{\tilde{w}(t+1)}{\|\tilde{w}(t+1)\|_2} \tag{5.2c}$$

In the above equations, t is the iteration number, $x(t)$ is the sample pattern chosen at iteration t uniformly at random from the patterns in the training set $\mathcal{X}$, and α_t is a small positive constant. Finally, $\Gamma(w) : \mathcal{R}^n \to \mathcal{R}^n = \nabla g(w)$ is the gradient of the penalty term for non-sparse solutions. This function has the interesting property that for very small values of $w_i(t)$, $\Gamma(w_i(t)) \simeq 2\sigma w_i(t)$. To see why, consider the i^{th} entry of the function $\Gamma(w(t)))$

$$\Gamma_i(w(t)) = \partial g(w(t))/\partial w_i(t) = 2\sigma_t w_i(t)(1 - \tanh^2(\sigma w_i(t)^2))$$

It is easy to see that $\Gamma_i(w(t)) \simeq 2\sigma w_i(t)$ for relatively small values of $w_i(t)$. And for larger values of $w_i(t)$, we get $\Gamma_i(w(t)) \simeq 0$ (see Figure 5.2). Therefore, by proper choice of

η and σ, equation (5.2b) suppresses small entries of $w(t)$ by pushing them towards zero, thus, favoring sparser results. To simplify the analysis, with some abuse of notation, we approximate the function $\Gamma_i(w(t), \theta_t)$ with the following function:

$$\Gamma_i(w(t), \theta_t) = \begin{cases} w_i(t) & \text{if } |w_i(t)| \leq \theta_t; \\ 0 & \text{otherwise,} \end{cases} \tag{5.3}$$

where θ_t is a small positive threshold.

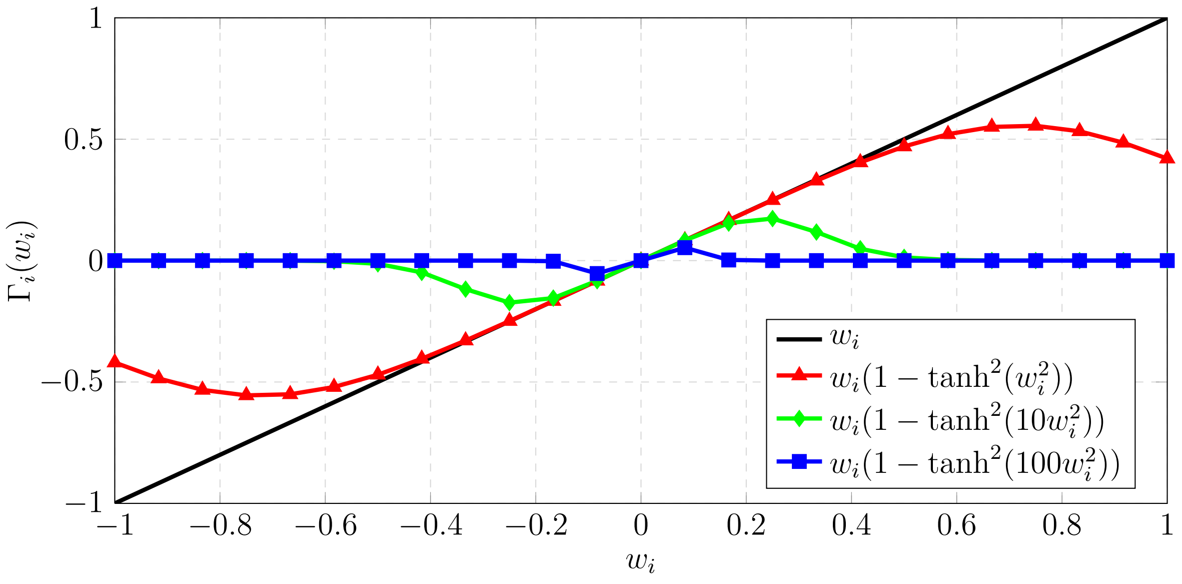


Figure 5.2: The sparsity penalty $\Gamma_i(w_i)$, which suppresses small values of the i^{th} entry of w in each iteration as a function of w_i and σ. The normalization constant 2σ has been omitted here to make comparison with function $f = w_i$ possible. As seen from the figure, the larger σ, the smaller its domain of effect will be (i.e., larger σ's are equivalent of smaller θ_t's is equation (5.3)).

Following the same approach as [36] and assuming α_t to be small enough such that equation (5.2c) can be expanded as powers of α_t, we can approximate equation (5.2) with the following simpler version:[2]

$$y(t) = x(t) \cdot w(t) \tag{5.4a}$$

[2]The expansion is basically equivalent to the deriving the Taylor expansion of $1/\sqrt{1+x^2}$ around 0 up to the second term.

Algorithm 1 Iterative Learning

```
**Input:** Set of patterns $x^\mu \in \mathcal{X}$ with $\mu = 1, \dots, C$, stopping point $\varepsilon$.
**Output:** $w$
  **while** $\sum_\mu |x^\mu \cdot w(t)|^2 > \varepsilon$ **do**
    Choose $x(t)$ at random from patterns in $\mathcal{X}$
    Compute $y(t) = x(t) \cdot w(t)$
    Update $w(t+1) = w(t) - \alpha_t y(t) \left( x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right) - \alpha_t \eta \Gamma(w(t), \theta_t)$.
    $t \leftarrow t + 1$.
  **end while**
```

$$w(t+1) = w(t) - \alpha_t \left(y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right) + \eta \Gamma(w(t), \theta_t) \right) \tag{5.4b}$$

In the above approximation, we also omitted the term $\alpha_t \eta \left(w(t) \cdot \Gamma(w(t), \theta_t) \right) w(t)$ since $w(t) \cdot \Gamma(w(t), \theta_t)$ would be negligible, specially as θ_t in equation (5.3) becomes smaller.

Remark 3. *In* practice*, and in order to ensure explicit sparsity, we choose η such that $\alpha_t \eta = 1$ for all t (so it would be more precise to use η_t instead of η). This way, and from equation* (5.3)*, we observe that in iteration $t+1$, the entries in $w(t)$ that are smaller than θ_t are set to zero so that $w(t+1)$ is calculated from a relatively sparse vector. Furthermore, at the end of the algorithm we apply the thresholding function again to ensure that values that are smaller than θ_t are set to zero.*

In general, in practice we could fix a small threshold and at the end of the learning phase set all values in the weight vector that are less than this threshold to zero. This way, although we might use a bit of precision (i.e., in the weight vector being exactly orthogonal to the patterns) but we gain a sparser neural graph which improves the performance in the recall phase. Nevertheless, the exact trade off between the precision and performance in the recall phase depends on the application and has to be obtained via fine-tuning.

The overall learning algorithm for one constraint node is given by Algorithm 1. In words, $y(t)$ is the projection of $x(t)$ on the basis vector $w(t)$. If for a given data vector $x(t)$, $y(t)$ is equal to zero, namely, the data is orthogonal to the current weight vector $w(t)$, then according to equation (5.4b) the weight vector will not be updated. However, if the data vector $x(t)$ has some projection over $w(t)$ then the weight vector is updated towards the direction to reduce this projection.

Since we are interested in finding m basis vectors, we have to do the above procedure *at least* m times (which can be performed in parallel).[3]

[3] In practice, we may have to repeat this process more than m times to ensure the existence of a set of m linearly independent vectors. However, our experimental results suggest that most of the time, repeating m

Remark 4. *Although we are interested in finding a sparse graph, note that too much sparseness is not desired. This is because we are going to use the feedback sent by the constraint nodes to eliminate input noise at pattern nodes during the recall phase. Now if the graph is too sparse, the number of feedback messages received by each pattern node is too small to be relied upon. Therefore, we must adjust the penalty coefficient η such that resulting neural graph is* sufficiently *sparse. In the section on experimental results, we compare the error correction performance for different choices of η.*

Convergence analysis

To prove the convergence of Algorithm 1, let $A = \mathbb{E}\{xx^T | x \in \mathcal{X}\}$ be the correlation matrix for the patterns in the training set. Also, let $A_t = x(t)(x(t))^\top$, (hence, $A = \mathbb{E}(A_t)$). Furthermore, let

$$E(t) = E(w(t)) = \frac{1}{C}\sum_{\mu=1}^{C}(w(t)^\top x^\mu)^2 \tag{5.5}$$

be the cost function. Finally, assume that the learning rate α_t is small enough so that terms that are $O(\alpha_t^2)$ can be neglected, similar to the approximation we made in deriving equation (5.4). In general, we pick the learning rate $\alpha_t \propto 1/t$ so that $\alpha_t > 0$, $\sum \alpha_t \to \infty$ and $\sum \alpha_t^2 < \infty$. We first show that the weight vector $w(t)$ never becomes zero, i.e., $\|w(t)\|_2 > 0$ for all t.

Lemma 5. *Assume we initialize the weights such that $\|w(0)\|_2 > 0$. Furthermore, assume $\alpha_t < \alpha_0 < 1/\eta$. Then, for all iterations t we have $\|w(t)\|_2 > 0$.*

Proof. We proceed by induction. To this end, assume $\|w(t)\|_2 > 0$ and let $w'(t) = w(t) - \alpha_t y(t)\left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2}\right)$. Note that $\|w'(t)\|_2^2 = \|w(t)\|_2^2 + \alpha_t^2 y(t)^2 \|x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2}\|_2^2 \geq \|w(t)\|_2^2 > 0$. Now,

$$\begin{aligned}
\|w(t+1)\|_2^2 &= \|w'(t)\|^2 + \alpha_t^2\eta^2\|\Gamma(w(t),\theta_t)\|^2 - 2\alpha_t\eta\Gamma(w(t),\theta_t)^\top w'(t) \\
&\geq \|w'(t)\|_2^2 + \alpha_t^2\eta^2\|\Gamma(w(t),\theta_t)\|^2 - 2\alpha_t\eta\|\Gamma(w(t),\theta_t)\|_2\|w'(t)\|_2 \\
&= \left(\|w'(t)\|_2 - \alpha_t\eta\|\Gamma(w(t),\theta_t)\|_2\right)^2
\end{aligned}$$

Thus, in order to have $\|w(t+1)\|_2 > 0$, we must have that $\|w'(t)\|_2 - \alpha_t\eta\|\Gamma(w(t),\theta_t)\|_2 > 0$. Given that, $\|\Gamma(w(t),\theta_t)\|_2 \leq \|w(t)\|_2 \leq \|w'(t)\|_2$, it is sufficient to have $\alpha_t\eta < 1$ in order to achieve the desired inequality. This proves the lemma. □

Next, the following theorem proves the convergence of Algorithm 1 to a minimum w^* such that $E(w^*) = 0$.

times would be sufficient.

Theorem 6. *Suppose the learning rate α_t is sufficiently small and both the learning rate α_t and the sparsity threshold θ_t decay according to $1/t$. Then, Algorithm 1 converges to a local minimum w^* for which $E(w^*) = 0$. At this point, w^* is orthogonal to the patterns in the data set $\mathcal{X}$.*

Proof. From equation (5.4b) recall that

$$w(t+1) = w(t) - \alpha_t \left(y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right) + \eta \Gamma(w(t), \theta_t) \right).$$

Let $Y(t) = \mathbb{E}_x(\mathcal{X} w(t))$, where $\mathbb{E}_x(.)$ is the expectation over the choice of pattern $x(t)$. Thus, we will have

$$Y(t+1) \quad = \quad Y(t) \left(1 + \alpha_t \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} \right) - \alpha_t \left(\mathcal{X} A w(t) + \eta \mathcal{X} \Gamma(w(t), \theta_t) \right).$$

Noting that $E(t) = \frac{1}{C}\|Y(t)\|_2^2$, we obtain

$$\begin{aligned}
E(t+1) &= E(t) \left(1 + \alpha_t \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} \right)^2 \\
&+ \frac{\alpha_t^2}{C} \| \mathcal{X} A w(t) + \eta \mathcal{X} \Gamma(w(t), \theta_t) \|_2^2 \\
&- 2\alpha_t \left(1 + \alpha_t \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} \right) \left(w(t)^\top A^2 w(t) \right) \\
&- 2\alpha_t \left(1 + \alpha_t \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} \right) \left(\eta w(t)^\top A \Gamma(w(t), \theta_t) \right) \\
&\overset{(a)}{\simeq} E(t) \left(1 + 2\alpha_t \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} \right) \\
&- 2\alpha_t \left(w(t)^\top A^2 w(t) + \eta w(t)^\top A \Gamma(w(t), \theta_t) \right) \\
&= E(t) - 2\alpha_t \left(w(t)^\top A^2 w(t) - \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} E(t) \right) \\
&- 2\alpha_t \eta w(t)^\top A \Gamma(w(t), \theta_t) \\
&\overset{(b)}{\simeq} E(t) - 2\alpha_t \left(w(t)^\top A^2 w(t) - \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} E(t) \right).
\end{aligned}$$

In the above equations, approximation (a) is obtained by omitting all terms that are $O(\alpha_t)^2$. Approximation (b) follows by noting that

$$\alpha_t \eta \|(w(t))^\top A \Gamma(w(t), \theta_t)\|_2 \leq \alpha_t \eta \|w(t)\|_2 \|A\|_2 \|\Gamma(w(t), \theta_t)\|_2 \leq \alpha_t \eta \|w(t)\|_2 \|A\|_2 (n\theta_t).$$

Now since $\theta_t = \Theta(\alpha_t)$, $\alpha_t \eta \|(w(t))^\top A \Gamma(w(t), \theta_t)\|_2 = O(\alpha_t^2)$ and, therefore, this term can be eliminated.

Therefore, in order to show that the algorithm converges, we need to show that

$$\left(w(t)^\top A^2 w(t) - \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} E(t) \right) \geq 0$$

to have $E(t+1) \leq E(t)$. Noting that $E(t) = w(t)^\top A w(t)$, we must show that $w^\top A^2 w \geq (w^\top A w)^2 / \|w\|_2^2$. Note that the left hand side is $\|Aw\|_2^2$. For the right hand side, we have

$$\frac{\|w^\top A w\|_2^2}{\|w\|_2^2} \leq \frac{\|w\|_2^2 \|Aw\|_2^2}{\|w\|_2^2} = \|Aw\|_2^2.$$

The above inequality shows that $E(t+1) \leq E(t)$, which implies that for sufficiently large number of iterations, the algorithm converges to a local minimum w^* where $E(w^*) = 0$. From Lemma 5 we know that $\|w^*\|_2 > 0$. Thus, the only solution for $E(w^*) = \|\mathcal{X} w^*\|_2^2 = 0$ would be to for w^* to be orthogonal to the patterns in the data set. □

Remark 7. *Note that the above theorem only proves that the obtained vector is orthogonal to the data set and says nothing about its degree of sparsity. The reason is that there is no guarantee that the dual basis of a subspace is sparse. The introduction of the penalty function $g(w)$ in problem (5.1) only encourages sparsity by suppressing small entries of w, i.e., shifting them towards zero if they are really small or leaving them intact if they are rather large. Our experimental results in section 5.6 show that in fact this strategy works perfectly and the learning algorithm results in sparse solutions.*

Making the Algorithm Parallel

In order to find m constraints, we need to repeat Algorithm 1 several times. Fortunately, we can repeat this process in parallel, which speeds up the algorithm and is more meaningful from a biological point of view as each constraint neuron can act independently of other neighbors. Although performing the algorithm in parallel may result in linearly dependent constraints once in a while, our experimental results show that starting from different random initial points, the algorithm converges to different distinct constraints most of the time. In addition, the chance of obtaining redundant constraints reduces if we start from a sparse random initial point. Besides, as long as we have "enough" distinct constraints[4], the recall

[4]Usually, the more constraints we learn, the better performance we get. However, as long as we learn a few constraints such that each pattern neuron is connected to at least a few (3 or more) constraint neurons, and no two pattern neurons have the same set of neighbors, then the number of constraints are "enough" to correct at least one error. A special case is the family of expander graphs discussed in Appendix 5.B.

algorithm in the next section can start eliminating noise and there is no need to learn all the distinct basis vectors of the null space defined by the training patterns (albeit the performance improves as we learn more and more linearly independent constraints). Therefore, we will use the parallel version to have a faster algorithm in the end.

5.4 Recall Phase

In the recall phase, we are going to design an iterative algorithm that corresponds to message passing on a graph. The algorithm exploits the fact that our learning algorithm resulted in the connectivity matrix of the neural graph which is sparse and orthogonal to the memorized patterns. Therefore, given a noisy version of the learned patterns, we can use the feedback from the constraint neurons in Fig. 5.1 to eliminate noise. More specifically, the linear input sums to the constraint neurons are given by the elements of the vector $W(x^\mu + z) = Wx^\mu + Wz = Wz$, with z being the integer-valued input noise. Based on observing the elements of Wz, each constraint neuron feeds back a message (containing info about z) to its neighboring pattern neurons. Based on this feedback, and exploiting the fact that W is sparse, the pattern neurons update their states in order to reduce the noise z.

It must also be mentioned that we initially assume *asymmetric* neural weights during the recall phase. More specifically, we assume the backward weight from constraint neuron i to pattern neuron j, denoted by W^b_{ij}, be equal to the sign of the weight from pattern neuron i to constraint neuron j, i.e., $W^b_{ij} = \text{sign}(W_{ij})$, where sign(x) is equal to $+1$, 0 or -1 if $x > 0$, $x = 0$ or $x < 0$, respectively. This assumption simplifies the error correction analysis. Later in section 5.4, we are going to consider another version of the algorithm which works with symmetric weights, i.e., $W^b_{ij} = W_{ij}$, and compare the performance of all suggested algorithms together in section 5.6.

Noise model

Here, we assume that the noise is integer valued and additive. Biologically speaking, the noise can be interpreted as a neuron skipping some spikes or firing more spikes than it should. It must be mentioned that neural states below 0 and above $Q - 1$ will be clipped to 0 and $Q - 1$, respectively. This is biologically justified as well since the firing rate of neurons can not exceed an upper bound and of course can not be less than zero.

The recall algorithms

The proposed algorithm for the recall phase comprises a series of *forward* and *backward* iterations. Two different methods are suggested here, which slightly differ from each other in the way pattern neurons are updated. The first one is based on the Winner-Take-All (WTA) approach and is given by Algorithm 2. The second approach, given by Algorithm 3, is simpler and is based on Majority Voting (MV) approach.

In both algorithms, first the pattern neurons send their states to the constraint neurons in the forward step (see Figure 5.3a). Each constraint neuron then calculates the weighted input sum over its incoming links and check if the the sum equals 0. Any value other than 0 indicates the presence of noise in at least one of its neighbors. The constraint neuron then informs its neighboring patten neurons about this incident by sending back the sign of the calculated input sum during the backward step (see Figure 5.3b).

At this point, each pattern neuron x_j computes

$$g_j^{(1)} = \frac{\sum_{i=1}^m W_{ij}^b y_i}{d_j}, \tag{5.6}$$

and

$$g_j^{(2)} = \frac{\sum_{i=1}^m |W_{ij}^b y_i|}{d_j}, \tag{5.7}$$

where d_j is the degree of pattern neuron j. The quantity $g_j^{(2)}$ can be interpreted as the number of feedback messages received by pattern neuron x_j from the constraint neurons. On the other hand, the sign of $g_j^{(1)}$ provides an indication of the sign of the noise that affects x_j, and $|g_j^{(1)}|$ indicates the confidence level in the decision regarding the sign of the noise (see Figure 5.3b).

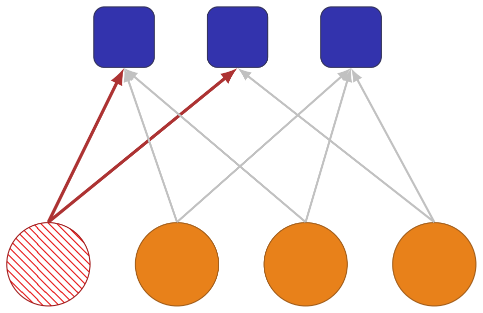
The pattern neurons now update their states based on $g_j^{(1)}$ and $g_j^{(2)}$. In the Winner-Take-All version,[5] only the pattern that has the maximum $g_j^{(2)}$ updates its state and the others keep their current values.[6] In the Majority-Voting method, however, any pattern neuron with $g_j^{(2)}$ larger than a threshold φ updates its state. In both cases, the amount of update is equal to $-g_j^{(1)}$.

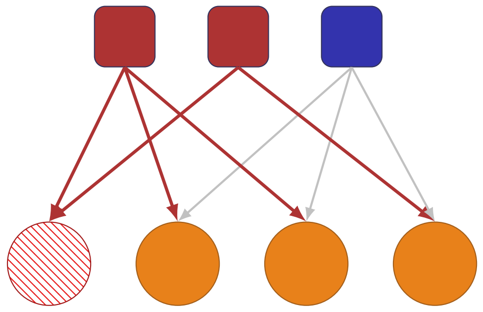
The entire WTA and MV algorithms are given in Algorithms 2 and 3, respectively.

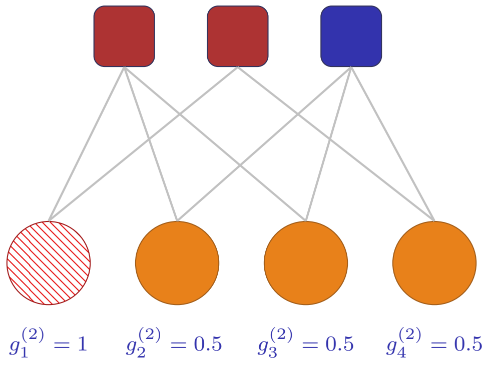
It is worthwhile mentioning that the Majority-Voting decoding algorithm is very similar to the Bit-Flipping algorithm of Sipser and Spielman to decode LDPC codes [49] and a similar approach in [50] for compressive sensing methods. Furthermore, both WTA and MV approaches are also similar to Gallager's Algorithms A and B for decoding Low Density Parity Check codes [19].

[5]The winner-take-all circuitry can be easily added to the neural model shown in Figure 5.1 using any of the classic WTA methods [13].

[6]Note that in order to maintain the current value of a neuron in case no input feedback is received, we can add self-loops to pattern neurons in Figure 5.1. These self-loops are not shown in the figure for clarity.


(a) Step 1 (forward iteration): Pattern neurons send their states to the constraint neurons. The hatched node is contaminated with noise.


(b) Step 2 (backward iteration): The constraint neurons check for violated constraints (dark red nodes) and send back proper messages to their neighbors.



(c) Step 3: Pattern neurons calculate the total (normalized) number of received feedback and the net value of input feedback.

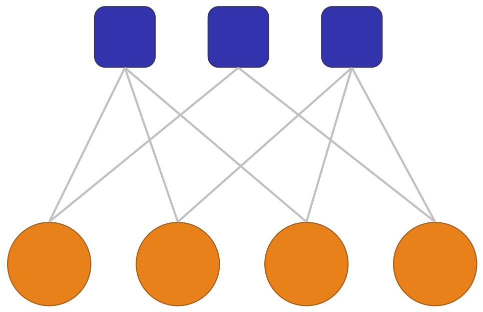

(d) Step 4: Pattern neurons with a large amount of input feedback (i.e., the first one) update their state and the error is eliminated.

Figure 5.3: The recall process.

Algorithm 2 Recall Algorithm: Winner-Take-All

```
Input: Connectivity matrix $W$, iteration $t_{\max}$
Output: $x_1, x_2, \ldots, x_n$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the weighted input sum $h_i = \sum_{j=1}^{n} W_{ij}x_j$, for each constraint neuron $y_i$ and set:
           $$y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise} \end{cases}.$$
3:     Backward iteration: Each neuron $x_j$ with degree $d_j$ computes
           $$g_j^{(1)} = \frac{\sum_{i=1}^{m} W_{ij}^b y_i}{d_j}, g_j^{(2)} = \frac{\sum_{i=1}^{m} |W_{ij}^b y_i|}{d_j}$$
4:     Find
           $$j^* = \arg\max_j g_j^{(2)}.$$
5:     Update the state of winner $j^*$: set $x_{j^*} = x_{j^*} - \text{sign}(g_{j^*}^{(1)})$.
6:     $t \leftarrow t + 1$
7: end for
```

Remark 8. *To give the reader some insight about why the neural graph should be sparse in order for the above algorithms to work, consider the backward iteration of both algorithms: it is based on counting the fraction of received input feedback messages from the neighbors of a pattern neuron. In the extreme case, if the neural graph is complete or dense, then a single noisy pattern neuron results in the violation of all or many constraint neurons in the forward iteration, as shown in Figure 5.4. Consequently, in the backward iteration even the non-corrupted pattern neurons receive feedback from their neighbors to update their state and it is impossible to tell which of the pattern neurons is the noisy one.*

However, if the graph is sparse, a single noisy pattern neuron only makes some of the constraints unsatisfied. Consequently, in the recall phase only the nodes which share the neighborhood of the noisy node receive input feedbacks and the fraction of the received feedbacks would be much larger for the original noisy node. Therefore, by merely looking at the fraction of received feedback from the constraint neurons, one can identify the noisy pattern neuron with high probability as long as the graph is sparse and the input noise is

Algorithm 3 Recall Algorithm: Majority-Voting

```
Input: Connectivity matrix $W$, threshold $\varphi$, iteration $t_{\max}$
Output: $x_1, x_2, \dots, x_n$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the weighted input sum $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each
       neuron $y_i$ and set:
           $y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise} \end{cases}.$
3:     Backward iteration: Each neuron $x_j$ with degree $d_j$ computes
           $g_j^{(1)} = \frac{\sum_{i=1}^{m} W_{ij}^b y_i}{d_j}, g_j^{(2)} = \frac{\sum_{i=1}^{m} |W_{ij}^b y_i|}{d_j}$
4:     Update the state of each pattern neuron $j$ according to $x_j = x_j - \text{sign}(g_j^{(1)})$ only if
       $|g_j^{(2)}| > \varphi$.
5:     $t \leftarrow t + 1$
6: end for
```

reasonably bounded.

Some Practical Modifications

Although algorithm 3 is fairly simple, each pattern neuron still needs two types of information: the number of received feedbacks and the net input sum. Although one can think of simple neural architectures to obtain the necessary information, we can modify the recall algorithm to make it more practical and simpler. The trick is to replace the degree of each node x_j with the ℓ_1-norm of the outgoing weights. In other words, instead of using $\|w_j\|_0 = d_j$, we use $\|w_j\|_1$. Furthermore, we assume symmetric weights, i.e., $W_{ij}^b = W_{ij}$.

Interestingly, in some of our experimental results corresponding to *denser graphs*, this approach performs better, as will be illustrated in section 5.6. One possible reason behind this improvement might be the fact that using the ℓ_1-norm instead of the ℓ_0-norm in Algorithm 3 will result in better differentiation between two vectors that have the same number of non-zero elements, i.e., have equal ℓ_0-norms, but differ from each other in the magnitude of the element, i.e., their ℓ_1-norms differ. Therefore, the network may use this additional information in order to identify the noisy nodes in each update of the recall algorithm.

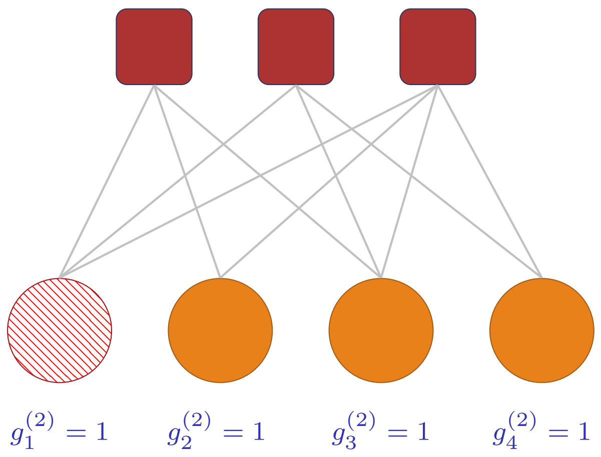


Figure 5.4: Dense neural graphs are not suitable for the proposed recall algorithms. Here, the hatched pattern neuron is contaminated with noise but all pattern neurons are getting high amounts of feedback, which makes it impossible to tell which one is the corrupted one.

Another practical modification could be modifying the algorithm to be able to deal with *real-valued* patterns. So far, our patterns are assumed to be integer-valued. However, in many practical situations in dealing with datasets of natural patterns, it might be difficult to map the patterns to be integer-valued via a quantization process, as some information is lost during the quantization. In such cases, we could slightly modify the proposed recall algorithms to cope with the task. Algorithm 4 shows the details of the proposed method. Basically, the algorithm is the same as Algorithm 3, with the following modifications:

1. The columns of matrix W are normalized to have a norm of 1.
2. In the denominator of g_j, $\|.\|_0$ is replaced by $\|.\|_2$.
3. Symmetric weight matrices are used, i.e., $W_{ij}^b = W_{ij}$.
4. The feedbacks sent by the constraint neurons are real-valued, i.e., $y_i = h_i = \sum_{j=1}^n W_{ij} x_j$.[7]

Performance Analysis

In this section, we derive a theoretical upper bound on the recall probability of error for the algorithms proposed in the previous section. To this end, we assume that the connectivity

[7]In practice, we only send feedback if $|h_i| > \psi$, with ψ being a small positive threshold.

Algorithm 4 Recall Algorithm: Real-Valued Patterns

```
Input: Connectivity matrix $W$ with normalized columns, threshold $\varphi$, iteration $t_{\max}$, small
    positive threshold $\psi$
Output: $x_1, x_2, \dots, x_n$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the weighted input sum $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each
       neuron $y_i$ and set $y_i = h_i$ if $|h_i| > \psi$.
3:     Backward iteration: Each neuron $x_j$ with degree $d_j$ computes
           $$g_j = \sum_{i=1}^{m} W_{ij} y_i$$
4:     Update the state of each pattern neuron $j$ according to $x_j = x_j - \text{sign}(g_j)$ only if
       $|g_j| > \varphi$.
5:     $t \leftarrow t + 1$
6: end for
```

graph W is sparse. With respect to this graph, we define the pattern and constraint degree distributions as follows.

Definition 9. *For the bipartite graph W, let Λ_i (Ω_j) denote the fraction of pattern (constraint) nodes of degree i (j). We call $\{\Lambda_1, \dots, \Lambda_m\}$ and $\{\Omega_1, \dots, \Omega_n\}$ the pattern and constraint degree distribution form the node perspective, respectively. Furthermore, it is convenient to define the degree distribution polynomials as*

$$\Lambda(z) = \sum_i \Lambda_i z^i \text{ and } \Omega(z) = \sum_i \Omega_i z^i.$$

The degree distributions are determined after the learning phase is finished and in this section we assume they are given. Furthermore, we consider an ensemble of random neural graphs with a given degree distribution and investigate the *average* performance of the recall algorithms over this ensemble. Here, the word "ensemble" refers to the fact that we assume having a number of *random* neural graphs with the given degree distributions and do the analysis for the average scenario. As such, the performance we achieve in practice could be better or worse than the average case analysis here (for more details on this issue, please see the discussions in Section 5.7 at the end of this chapter).

To simplify analysis, we assume that the noise entries are ± 1. However, the proposed recall algorithms can work with any integer-valued noise and our experimental results suggest that this assumption is not necessary in practice.

Finally, we assume that the errors do not cancel each other out in the constraint neurons (as long as the number of errors is fairly bounded). This is in fact a realistic assumption because the neural graph is weighted, with weights belonging to the real field, and the noise values are integers. Thus, the probability that the weighted sum of some integers be equal to zero is negligible.

We do the analysis only for the Majority-Voting algorithms since if we choose the Majority-Voting update threshold $\varphi = 1$, roughly speaking, we will have the winner-take-all algorithm.[8]

To start the analysis, let m be the total number of constraint neurons and $\mathcal{E}_t$ denote the set of erroneous pattern nodes at iteration t, and $\mathcal{N}(\mathcal{E}_t)$ be the set of constraint nodes that are connected to the nodes in $\mathcal{E}_t$, i.e., these are the constraint nodes that have at least one neighbor in $\mathcal{E}_t$. In addition, let $\mathcal{N}^c(\mathcal{E}_t)$ denote the (complementary) set of constraint neurons that do not have any connection to any node in $\mathcal{E}_t$. Denote also the average neighborhood size of $\mathcal{E}_t$ by $S_t = \mathbb{E}(|\mathcal{N}(\mathcal{E}_t)|)$. Finally, let $\mathcal{C}_t$ be the set of correct pattern nodes.

Before getting to the upper bound, we need to establish a relationship between the average number of violated constraints in iteration t, S_t, and the number of corrupted pattern nodes, $|\mathcal{E}_t|$. The following lemma provides us with such a relationship.

Lemma 10. *The average neighborhood size S_t in iteration t is given by:*

$$S_t = m\left(1 - \left(1 - \frac{\bar{d}}{m}\right)^{|\mathcal{E}_t|}\right) \tag{5.8}$$

where $\bar{d}$ is the average degree for pattern nodes.

Proof. The proof is given in Appendix 5.A. □

Now based on the error correcting algorithm and the above notations, in a given iteration two types of error events are possible:

1. Type-1 error event: A node $x \in \mathcal{C}_t$ decides to update its value. The probability of this event is denoted by $P_{e_1}(t)$.

2. Type-2 error event: A node $x \in \mathcal{E}_t$ updates its value in the wrong direction. Let $P_{e_2}(t)$ denote the probability of an error for this type.

[8]It must be mentioned that choosing $\varphi = 1$ does not yield the WTA algorithm exactly because in the original WTA, only one node is updated in each round. However, in this version with $\varphi = 1$, all nodes that receive feedback from all their neighbors are updated. Nevertheless, the performance of the both algorithms should be rather similar.

We start the analysis by finding explicit expressions and upper bounds on the average of $P_{e_1}(t)$ and $P_{e_2}(t)$ over all nodes as a function S_t. The following two lemmas give us the results we need.

Lemma 11. *For $\varphi \to 1$, the probability that a non-corrupted pattern neuron decides to (mistakenly) update its state is given by*

$$P_{e_1}(t) = \Lambda\left(\frac{S_t}{m}\right).$$

Proof. To begin with, let $P_1^x(t)$ be the probability that a node $x \in \mathcal{C}_t$ with degree d_x updates its state. We have:

$$P_1^x(t) = \Pr\left\{\frac{|\mathcal{N}(\mathcal{E}_t) \cap \mathcal{N}(x)|}{d_x} \geq \varphi\right\} \tag{5.9}$$

where $\mathcal{N}(x)$ is the neighborhood of x. Assuming random construction of the graph and relatively large graph sizes, one can approximate $P_1^x(t)$ by

$$P_1^x(t) \approx \sum_{i=\lceil \varphi d_x \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t}{m}\right)^i \left(1 - \frac{S_t}{m}\right)^{d_x - i}, \tag{5.10}$$

where the approximation is derived calculating the probability that x shares more than $\lceil \varphi d_x \rceil$ neighbors with the nodes in $\mathcal{N}(\mathcal{E}_t)$. In the above equation, S_t/m represents the probability of having one of the d_x edges connected to the S_t constraint neurons that are neighbors of the erroneous pattern neurons.

As a result of the above equations, we have:

$$P_{e_1}(t) = \mathbb{E}_{d_x}(P_1^x(t)), \tag{5.11}$$

where $\mathbb{E}_{d_x}$ denote the expectation over the degree distribution $\{\Lambda_1, \ldots, \Lambda_m\}$.

Note that if $\varphi = 1$, the above equation simplifies to

$$P_{e_1}(t) = \Lambda\left(\frac{S_t}{m}\right).$$

□

Lemma 12. *Let $\mathcal{E}_t^* = \mathcal{E}_t \setminus \{x\}$ be the neighborhood size of all corrupted pattern neurons except a particular neuron x and $S_t^* = \mathbb{E}(|\mathcal{N}(\mathcal{E}_t^*)|)$. Then the probability that a noisy pattern neuron updates its state in the wrong direction is upper bounded by $P_{e_2}(t) \leq \mathbb{E}_{d_x}(P_2^x(t))$, where $P_2^x(t)$ is given by*

$$P_2^x(t) = \sum_{i=\lceil d_x/2 \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t^*}{m}\right)^i \left(1 - \frac{S_t^*}{m}\right)^{d_x - i}. \tag{5.12}$$

Proof. A node $x \in \mathcal{E}_t$ makes a wrong decision if the net input sum it receives has a different sign than the sign of noise it experiences. Instead of finding an exact relation, we bound this probability by the probability that the neuron x shares at least half of its neighbors with other neurons. Otherwise, since the number of unique neighbors of this noisy neuron is larger than half of its total neighbors, we are sure that the overall feedback on the direction of update will be correct.

Thus, we obtain the upper bound $P_{e_2}(t) \leq \text{Pr}\{\frac{|\mathcal{N}(\mathcal{E}_t^*) \cap \mathcal{N}(x)|}{d_x} \geq 1/2\}$, where $\mathcal{E}_t^* = \mathcal{E}_t \setminus x$. Letting $P_2^x(t) = \text{Pr}\{\frac{|\mathcal{N}(\mathcal{E}_t^*) \cap \mathcal{N}(x)|}{d_x} \geq 1/2 | \text{deg}(x) = d_x\}$, we will have:

$$P_2^x(t) = \sum_{i=\lceil d_x/2 \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t^*}{m}\right)^i \left(1 - \frac{S_t^*}{m}\right)^{d_x - i}$$

where $S_t^* = \mathbb{E}(|\mathcal{N}(\mathcal{E}_t^*)|)$

Therefore, we will have:

$$P_{e_2}(t) \leq \mathbb{E}_{d_x}(P_2^x(t)). \tag{5.13}$$

□

Finally, by combining the results of Lemmas 11 and 12, the following theorem provides an upper bound on the final recall probability of error, P_E.

Theorem 13. *Let $\bar{P}_i^x = \mathbb{E}_{d_x}\{P_i^x\}$, where P_i^x for $i = 1, 2$ is given by equations* (5.10) *and* (5.12). *Furthermore, let $|\mathcal{E}_0|$ be the initial number of noisy nodes in the input pattern. Then, the overall recall error probability, P_E, is upper bounded by*

$$P_E \leq 1 - \left(1 - \frac{n - |\mathcal{E}_0|}{n}\bar{P}_1^x - \frac{|\mathcal{E}_0|}{n}\bar{P}_2^x\right)^n. \tag{5.14}$$

Proof. Based on the results of lemmas 11 and 12 the bit error probability at iteration t would be

$$\begin{aligned} P_b(t+1) &= \text{Pr}\{x \in \mathcal{C}_t\}P_{e_1}(t) + \text{Pr}\{x \in \mathcal{E}_t\}P_{e_2}(t) \\ &= \frac{n - |\mathcal{E}_t|}{n}P_{e_1}(t) + \frac{|\mathcal{E}_t|}{n}P_{e_2}(t). \end{aligned} \tag{5.15}$$

Therefore, the average block error rate is given by the probability that at least one pattern node x is in error. Therefore:

$$P_e(t) = 1 - (1 - P_b(t))^n. \tag{5.16}$$

Equation (5.16) gives the probability of making a mistake in iteration t. Therefore, we can bound the overall probability of error, P_E, by setting $P_E = \lim_{t \to \infty} P_e(t)$. To this end, we have to recursively update $P_b(t)$ in equation (5.15) and using $|\mathcal{E}_{t+1}| \approx nP_b(t+1)$. However, since we have assumed that the noise values are ± 1, we can provide an upper bound on the total probability of error by considering

$$P_E \leq P_e(1). \tag{5.17}$$

In other words, we assume that the recall algorithms either corrects the input error in the first iteration or an error is declared.[9] As the initial number of noisy nodes grow, the above bound becomes tight. Thus, in summary we have:

$$P_E \leq 1 - \left(1 - \frac{n - |\mathcal{E}_0|}{n}\bar{P}_1^x - \frac{|\mathcal{E}_0|}{n}\bar{P}_2^x\right)^n$$

where $\bar{P}_i^x = \mathbb{E}_{d_x}\{P_i^x\}$ and $|\mathcal{E}_0|$ is the number of noisy nodes in the input pattern initially. $\square$

Remark 14. *One might hope to further simplify the above inequalities by finding closed form approximation of equations (5.10) and (5.12). However, as one expects, this approach leads to very loose and trivial bounds in many cases. Therefore, in our experiments shown in section 5.6 we compare simulation results to the theoretical bound derived using equations (5.10) and (5.12).*

The following corollary derives a lower bound on the probability of correcting a single input error, which will become handy in the upcoming chapters to improve the recall algorithm.

Corollary 15. *Let $\bar{d}$ be the average degree of pattern neurons in the neural graph. Then, if the neural graph is constructed randomly, Algorithm 3 can correct (at least) a single error with probability at least $\left(1 - \Lambda\left(\frac{\bar{d}}{m}\right)\right)^n$ when $\varphi \to 1$.*

Proof. In the case of a single error, we are sure that the corrupted node will always be updated towards the correct direction. Thus, $\bar{P}_2^x = 0$. Furthermore, since we have only a single erroneous pattern neuron initially, $|\mathcal{E}_0| = 1$. Consequently, from equation (5.14) we have

$$P_c = 1 - P_E \geq \left(1 - \frac{n-1}{n}\bar{P}_1^x\right)^n > \left(1 - \bar{P}_1^x\right)^n$$

Now from Lemma 11 we know that when $\varphi \to 1$, we have $\bar{P}_1^x = \Lambda(S_0/m)$. On the other hand, S_0 is the average neighborhood size of noisy neurons in the first iteration. Since we

[9]Obviously, this bound is not tight in practice and one might be able to correct errors in later iterations. In fact simulation results confirm this expectation. However, this approach provides a nice analytical upper bound since it only depends on the initial number of noisy nodes.

only have one noisy pattern neuron initially, we have $S_0 = \bar{d}$. Combining all these together we obtain

$$P_c > \left(1 - \Lambda\left(\frac{\bar{d}}{m}\right)\right)^n,$$

which proves the corollary. □

Remark 16. *The expression in the above lemma can be further simplified to*

$$P_c > \left(1 - \Lambda\left(\frac{\bar{d}}{m}\right)\right)^n \geq \left(1 - \left(\frac{\bar{d}}{m}\right)^{d_{\min}}\right)^n,$$

where $d_{\min}$ is the minimum degree of pattern neurons in the neural graph. This shows the significance of having high-degree pattern neurons. For instance, in the extreme case of $d_{\min} = 0$, which would happen if the redundancy is not distributed uniformly among the pattern neurons (see the discussion in in Section 5.7), then we obtain the trivial bound of $P_c \geq (1 - \Lambda_0)^n$, where Λ_0 is the fraction of pattern neurons with degree equal to 0. On the limit of large n we get $P_c \geq (1-1/n)^n \to 1/e^{\Lambda_0 n}$. Thus, even if only a single pattern neuron has a zero degree, probability of correcting a single error drops significantly.

While Corollary 15 provides a lower bound on the probability of correcting a single error when the connectivity graph is assumed to be constructed randomly, the following corollary shows that when some very mild conditions are satisfied, then the recall algorithms will correct a single input error with probability 1.

Corollary 17. *If no two pattern neurons share the exact same neighborhood in the neural graph, then for $\varphi \to 1$, Algorithms 2 and 3 can correct (at least) a single error.*

Proof. We prove the corollary for Algorithm 3. The proof for Algorithm 2 would be very similar.

Without loss of generality, suppose the first pattern neuron is contaminated by a $+1$ external error, i.e., $z = [1, 0, \dots, 0]$. As a result

$$y = \text{sign}\left(W(x+z)\right) = \text{sign}\left(Wx + Wz\right) = \text{sign}\left(Wz\right) = \text{sign}\left(W_1\right),$$

where W_i is the i$^{\text{th}}$ column of W. Then, the feedback transmitted by the constraint neurons is sign(W_1). As a result, decision parameters of pattern neuron i, i.e., $g_i^{(2)}$ in Algorithm 3 will be

$$g_i^{(2)} = \frac{\langle \text{sign}(W_i), \text{sign}(W_1)\rangle}{d_i} = \begin{cases} 1, & i = 1 \\ < 1, & i > 1 \end{cases},$$

where the inequality follows since no two neurons share the same neighborhood. As a result, for $\varphi \to 1$, only the first neuron, the noisy one, will update its value towards the correct state, thus, resulting in the correction of the single error. This proves the corollary. □

Remark 18. *As mentioned earlier, in this chapter we will perform the analysis for general sparse bipartite graphs. However, restricting ourselves to a particular type of sparse graphs known as "expanders" allows us to prove stronger results on the recall error probabilities. More details can be found in Appendix 5.C. Since it is very difficult, if not impossible in certain cases, to make a graph an expander during an iterative learning method, we have focused on the more general case of sparse neural graphs.*

5.5 Pattern Retrieval Capacity

So far, we have proposed a simple iterative algorithm to memorize the patterns, and a decoding algorithm to deal with noise in the recall phase. It is now time to prove that the whole model is capable of memorizing an exponential number of patterns.

It is interesting to see that, except for its obvious influence on the learning time, the number of patterns C does not have any effect in the learning or recall algorithm. As long as the patterns belong to a subspace, the learning algorithm will yield a matrix which is orthogonal to all of the patterns in the training set. Furthermore, in the recall phase, all we deal with is Wz, with z being the noise which is independent of the patterns.

Therefore, in order to show that the pattern retrieval capacity is exponential in n, all we need to show is that there exists a "valid" training set $\mathcal{X}$ with C patterns of length n for which $C \propto a^{rn}$, for some $a > 1$ and $0 < r$. By valid we mean that the patterns should belong to a subspace with dimension $k < n$ and the entries in the patterns should be non-negative integers. The next theorem proves the desired result.

Theorem 19. *Let $a > 1$ be a real number, and $Q > 2$ be an integer. Let n be an integer and $k = \lfloor rn \rfloor$ for some $0 < r < 1$. Then, there exists a set of $C = a^{\lfloor rn \rfloor}$ vectors in $\{0, 1, \dots, Q-1\}^n$ such that the dimension of the vector space spanned by these vectors is k. This set can be memorized by the proposed learning algorithm.*

Proof. The proof is based on construction: we construct a data set $\mathcal{X}$ with the required properties. To start, consider a matrix $G \in \mathbb{R}^{k\times n}$ with rank k and $k = \lfloor rn \rfloor$, with $0 < r < 1$. Let the entries of G be non-negative integers, between 0 and $\gamma - 1$, with $\gamma \geq 2$. Furthermore, assume G is constructed randomly, i.e., each entry in column j of G is non-zero with some probability d_j/k, where d_j is the average non-zero entries we would like to have in column j of G.

We start by constructing the patterns in the data set as follows: consider a set of random vectors $u^\mu \in \mathbb{R}^k$, $\mu = 1, \dots, C$, with integer-valued entries between 0 and $\upsilon - 1$, where $\upsilon \geq 2$. We set the pattern $x^\mu \in \mathcal{X}$ to be $x^\mu = u^\mu \cdot G$, *if* all the entries of x^μ are between 0 and $Q - 1$. Obviously, since both u^μ and G have only non-negative entries, all entries in

x^μ are non-negative. Therefore, we just need to ensure that entries of x^μ are between 0 and $Q-1$.

The j^{th} entry in x^μ is equal to $x_j^\mu = u^\mu \cdot G_j$, where G_j is the j^{th} column of G. Suppose G_j has d_j non-zero elements. Then, we have:

$$x_j^\mu = u^\mu \cdot G_j \leq d_j(\gamma-1)(\upsilon-1).$$

Therefore, if $d^* = \max_j d_j$, we can choose γ, υ and d^* such that

$$Q - 1 \geq d^*(\gamma-1)(\upsilon-1) \tag{5.18}$$

to ensure that all entries of x^μ are less than Q.

As a result, since there are υ^k vectors u with integer entries between 0 and $\upsilon - 1$, we will have $\upsilon^k = \upsilon^{rn}$ patterns forming $\mathcal{X}$. This means $C = \upsilon^{\lfloor rn \rfloor}$, which is exponential in n if $\upsilon \geq 2$. $\square$

As an example, if G can be selected to be a sparse 200×400 matrix with $0/1$ entries (i.e., $\gamma = 2$) and $d^* = 10$, and u is also chosen to be a vector with $0/1$ elements (i.e., $\upsilon = 2$), then it is sufficient to choose $Q \geq 11$ to have a pattern retrieval capacity of $C = 2^{rn}$.

Remark 20. *Note that inequality (8.5) was obtained for the worst-case scenario and in fact is very loose. Therefore, even if it does not hold, we may still be able to memorize a very large number of patterns since a big portion of the generated vectors x^μ will have entries less than Q. These vectors correspond to the message vectors u^μ that are "sparse" as well, i.e., do not have all entries greater than zero. The number of such vectors is polynomial in n, the degree of which depends on the number of non-zero entries in u^μ.*

5.6 Simulation Results

This section provides experimental results to investigate different aspects of both learning and recall algorithms.[10]

Simulation Scenario

We have simulated the proposed learning and recall algorithms for three different network sizes $n = 200, 400, 800$, with $k = n/2$ for all cases. For each case, we considered a few different setups with different values for α, η, and θ in the learning algorithm 1, and different

[10]The MATLAB codes that are used in conducting the simulations mentioned in this chapter are available online at `https://github.com/saloot/NeuralAssociativeMemory`.

φ for the Majority-Voting recall algorithm 3. For brevity, we do not report all the results for various combinations but present only a selection of them to give insight on the performance of the proposed algorithms.

In all cases, we generated 50 random training sets using the approach explained in the proof of Theorem 19, i.e., we generated a generator matrix G at random with $0/1$ entries and $d^* = 10$. We also used $0/1$ generating message words u and put $Q = 11$ to ensure the validity of the generated training set.

However, since in this setup we will have 2^k patterns to memorize, doing a simulation over all of them would take a long time. Therefore, we have selected a random sample subset $\mathcal{X}$ each time with size $C = 10^5$ for each of the 50 generated sets and used these subsets as the training set.

For each setup, we performed the learning algorithm and then investigated the average sparsity of the learned constraints over the ensemble of 50 instances. As explained earlier, all the constraints for each network were learned in parallel, i.e., to obtain $m = n - k$ constraints, we executed Algorithm 1 from random initial points m time.

As for the recall algorithms, the error correcting performance was assessed for each setup, averaged over the ensemble of 50 instances. The empirical results are compared to the theoretical bounds derived in Section 5.4 as well.

Learning Phase Results

In the learning algorithm, we pick a pattern from the training set each time and adjust the weights according to Algorithm 1. Once we have gone over all the patterns, we repeat this operation several times to make sure that update for one pattern does not adversely affect the other learned patterns. Let t be the iteration number of the learning algorithm, i.e., the number of times we have gone over the training set so far. Then we set $\alpha_t \propto \alpha_0/t$ to ensure that the conditions of Theorem 6 are satisfied. Interestingly, all of the constraints converged in at most two learning iterations for all different setups. Therefore, the learning is very fast in this case.

Figure 5.5 illustrates the Mean Square Error (MSE), defined by equation 5.5, per iteration of the learning algorithm for $n = 400$, $k = 200$ and for different values of α_0 and θ_0. In all cases we have the sparsity penalty coefficient η set to 1. In the figure, we first notice the effect of increasing the learning rate α_0 in speeding up the learning process. This behavior is expected. However, it must be noted that increasing α_0 beyond a certain limit might cause stability issues and even result in divergence of the algorithm.

The second trend that we observe in Figure 5.5 is that for a fixed value of α_0, increasing θ_0 results in faster convergence (the dashed vs. dotted curves), i.e., enforcing higher sparsity improves the convergence speed. However, as we will see shortly, there is again an upper limit on the extent to which we can increase θ_0 as very high values could also make the whole algorithm diverge.

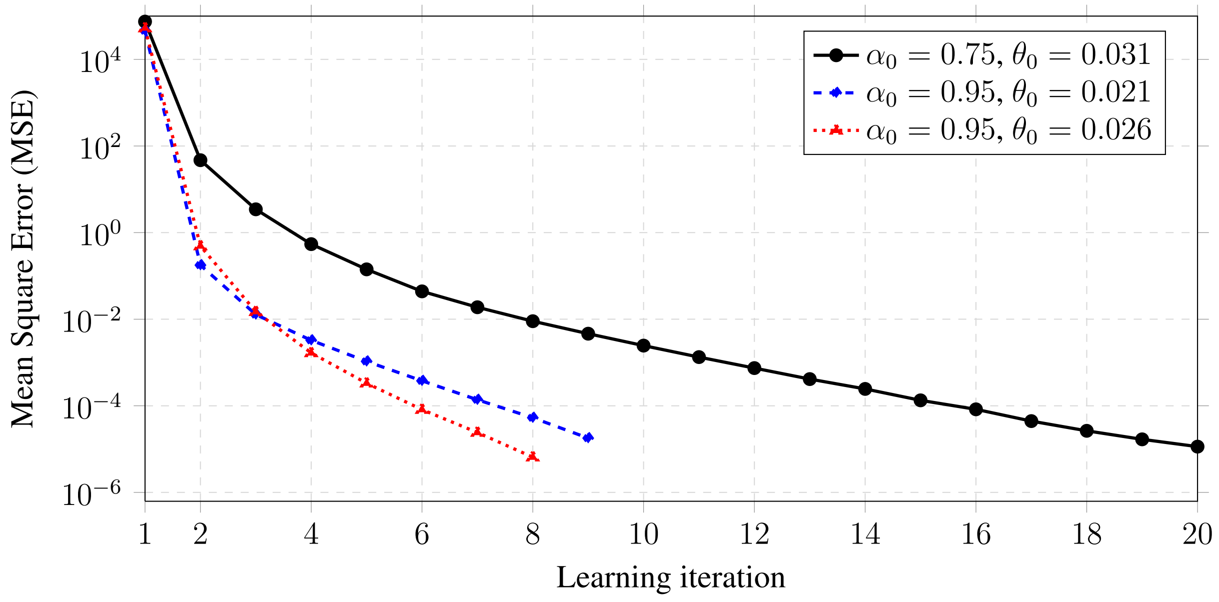


Figure 5.5: The MSE per iteration in the proposed learning algorithm for $n = 400$, $k = 200$ and different values of α_0 and θ_0. Note how increasing learning rate α_0 speeds up the learning process.

Figure 5.6 illustrates the same results but for $n = 200$, $k = 100$. Here, $\alpha_0 = 0.75$ and $\eta = 0.45$ for both cases. However, θ_0 is varied. One clearly sees that once θ_0 is too high (the blue curve), the algorithm takes much longer to converge as a lot of entries in w (below θ_0) are set to zero in each iteration, which as θ_0 increases is equivalent to larger amounts of information loss. Therefore, the learning process takes longer to finish in such a case.

Figure 5.7 illustrates the percentage of pattern nodes with the specified sparsity degree defined as $\varrho = \kappa/n$, where κ is the number of non-zero elements. From the figure we notice two trends. The first is the effect of sparsity threshold, which as it is increased, the network becomes sparser. The second one is the effect of network size, which as it grows, the connections become sparser.

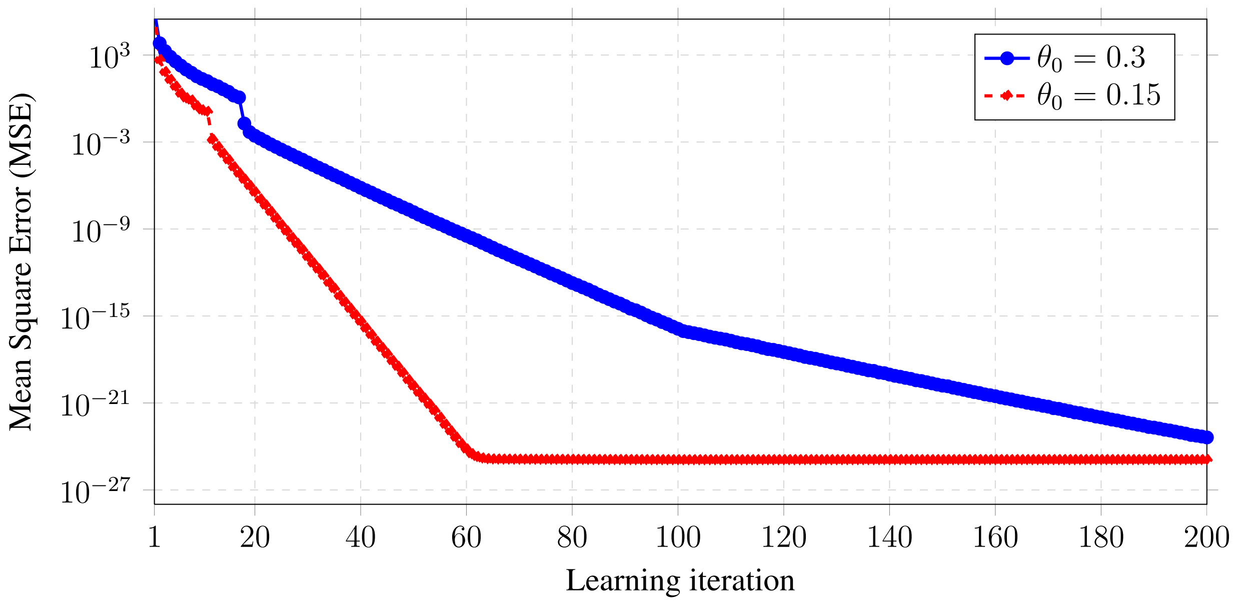


Figure 5.6: The MSE per iteration in the proposed learning algorithm for $n = 200$, $k = 100$ and different values of θ_0. Here, $\alpha_0 = 0.75$ and $\eta = 0.45$ for both cases. Note how very large values of sparsity threshold (θ_0) slows than the learning process.

Recall Phase Results

For the recall phase, in each trial we pick a pattern randomly from the training set, corrupt a given number of its symbols with ± 1 noise and use the suggested algorithm to correct the errors.[11] A pattern error is declared if the output does not match the correct pattern. We compare the performance of the two recall algorithms: Winner-Take-All (WTA) and Majority-Voting (MV). Table 5.1 shows the simulation parameters in the recall phase for all scenarios (unless specified otherwise).

Figure 5.8 illustrates the effect of the sparsity threshold θ on the performance of the error correcting algorithm in the recall phase. Here, we have $n = 400$ and $k = 200$. Two different sparsity thresholds are compared, namely $\theta_t \propto 0.031/t$ and $\theta_t \propto 0.021/t$. Clearly, as the network becomes sparser, i.e., as θ increases, the performance of both recall algorithms improve.

[11]Note that the ± 1 noise values are considered for simplicity and the algorithm can work with other integer-valued noise models as well.

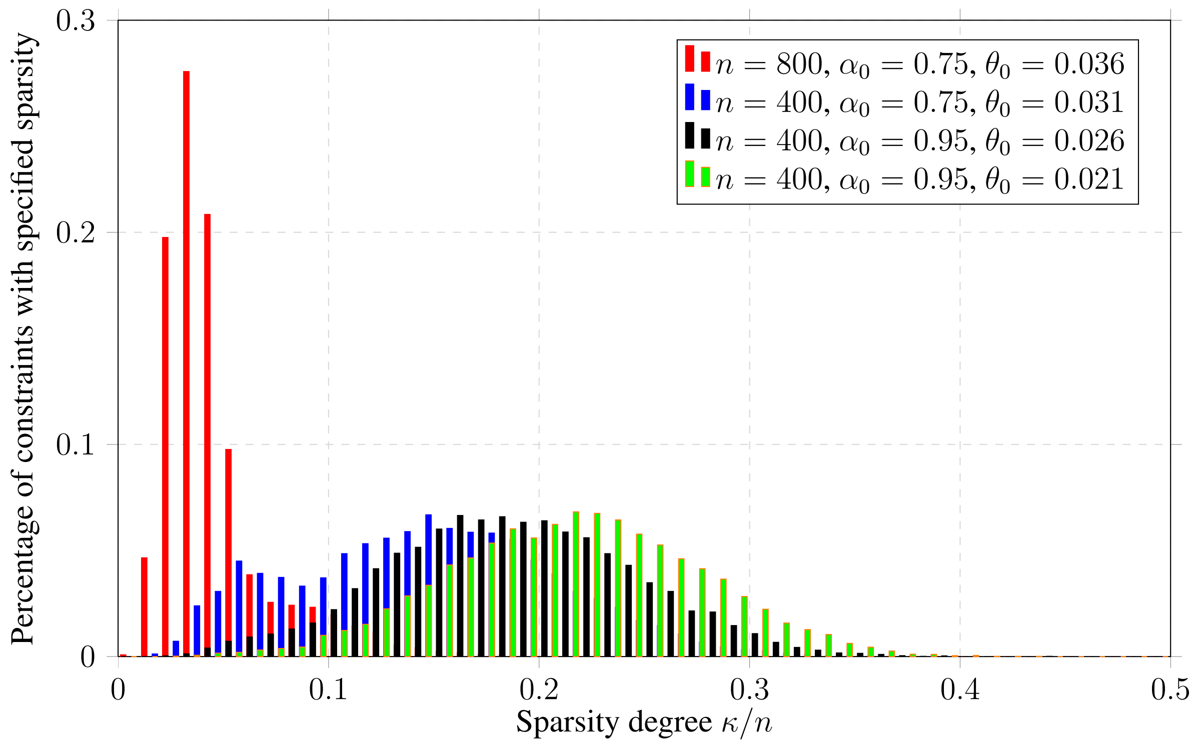


Figure 5.7: The percentage of variable nodes with the specified sparsity degree and different values of network sizes and sparsity thresholds. The sparsity degree is defined as $\varrho = \kappa/n$, where κ is the number of non-zero elements.

Table 5.1: Simulation parameters

Parameter	φ	$t_{\max}$	ε	η
Value	1	$20\|z\|_0$	0.001	$1/\alpha_t$

The effect of the sparsity penalty coefficient η is investigated in Figure 5.9. In this case, we have $n = 200$ and $k = 100$. We compare the effect of two different values of η, namely $\eta = 0.45$ and 0.85. The sparsity threshold θ_0 is the same for the two cases and is equal to 0.3. Note how increasing η results in a slightly better performance, which might be due to the fact that larger η's is equivalent to higher sparsity, which as we saw earlier translates into

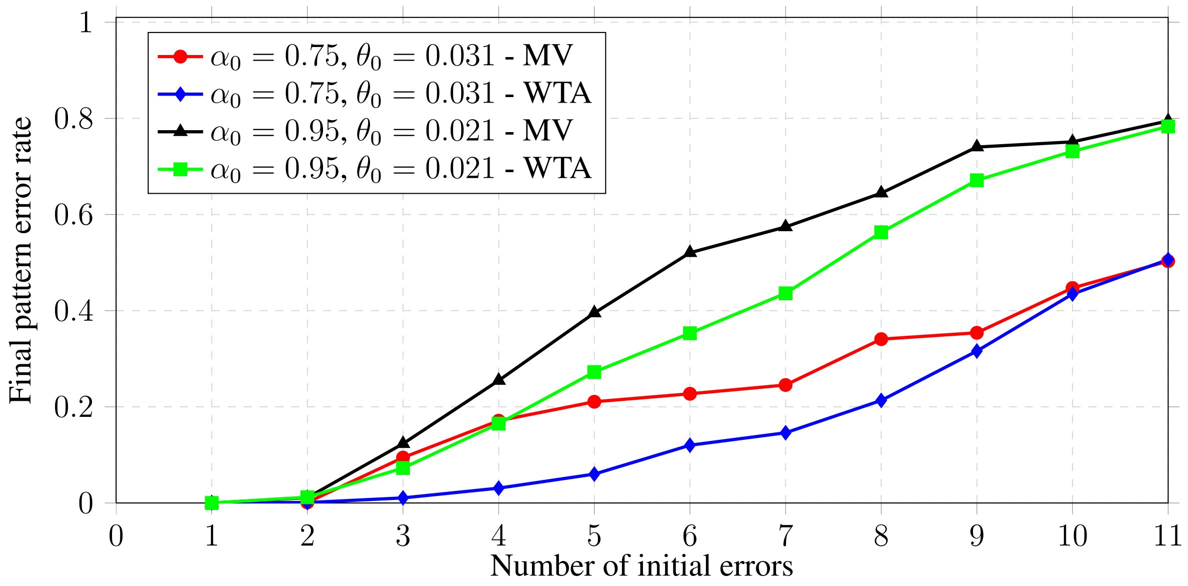


Figure 5.8: Pattern error rate against the initial number of erroneous nodes for two different values of θ_0. Here, the network size is $n = 400$ and $k = 200$. The blue and the red curves correspond to the sparser network (larger θ_0) and clearly show a better performance.

lower recall error rates.

In Figure 5.10 we have investigated the effect of network size on the performance of recall algorithms by comparing the pattern error rates for two different network size, namely $n = 800$ and $n = 400$ with $k = n/2$ in both cases. As obvious from the figure, the performance improves to a great extent when we have a larger network. This is partially because of the fact that in larger networks, the connections are relatively sparser as well.

Figure 5.11 compares the results obtained in simulation with the upper bound derived in Section 5.4. Note that as expected, the bound is quite loose since in deriving inequality (5.16) we only considered the first iteration of the algorithm.

We also investigate the performance of the modified more practical version of the Majority-Voting algorithm, which was explained in Section 5.4. Figure 5.12 compares the performance of the WTA and original MV algorithms with the modified version of MV algorithm for a network with size $n = 200$, $k = 100$ and learning parameters $\alpha_t \propto 0.45/t$, $\eta = 0.45$ and $\theta_t \propto 0.015/t$. The neural graph of this particular example is rather dense, because of small n and sparsity threshold θ. Therefore, here the modified version of the Majority-Voting

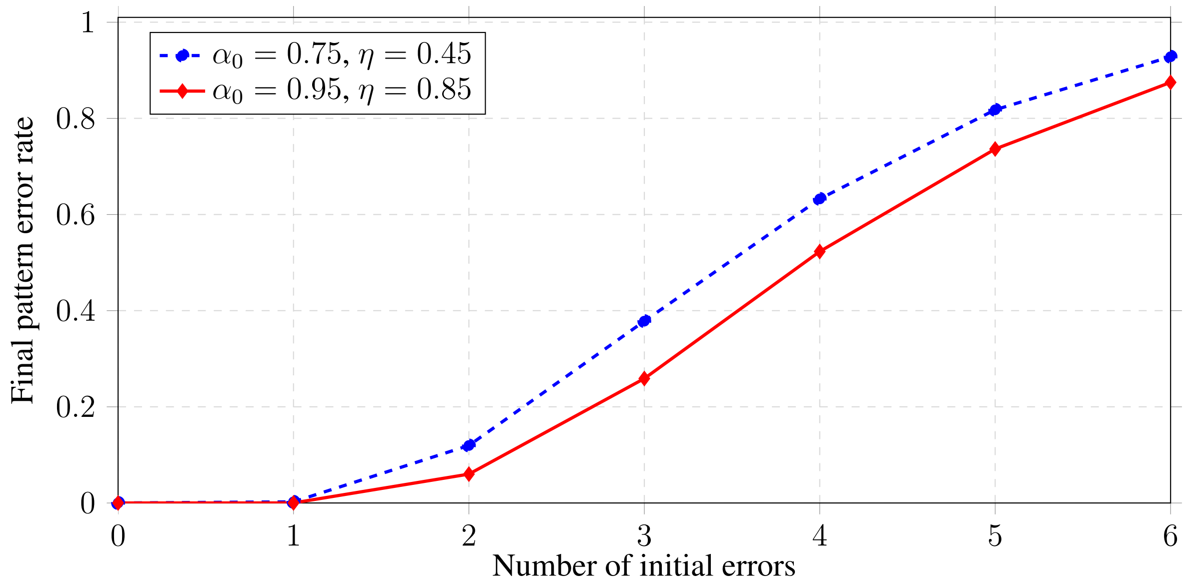


Figure 5.9: Pattern error rate against the initial number of erroneous nodes could achieve vert f η. Here, the network size is $n = 200$ and $k = 100$. The sparsity threshold is the same for the two cases and equals $\theta_0 = 0.3$.

algorithm performs better because of the extra information provided by the ℓ_1-norm (rather than the ℓ_0-norm in the original version of the Majority-Voting algorithm). However, note that we did not observe this trend for the other simulation scenarios where the neural graph was sparser.

Finally, Figure 5.13 compares the final recall error probability of the proposed method in this chapter and the multi-state complex-valued neural networks proposed in [38] and [40].[12] To have a fair comparison, we applied the methods of [38] and [40] to a dataset of randomly generated patterns, which is the setting that these methods are designed for, to see how the results compare to those returned by our method in its natural setting, i.e., patterns belonging to a subspace. As seen from Fig. 5.13 the proposed method in this chapter achieves very small recall errors while having a much larger pattern retrieval capacity at the same time. We then applied the methods of [38] and [40] to our dataset, where patterns belong to a subspace

[12]The learning method for the multi-state network proposed in [39], as also mentioned by the authors, is computationally prohibitive for the rather large network sizes we are interested here. As a result, it couldn't be considered for this chapter.

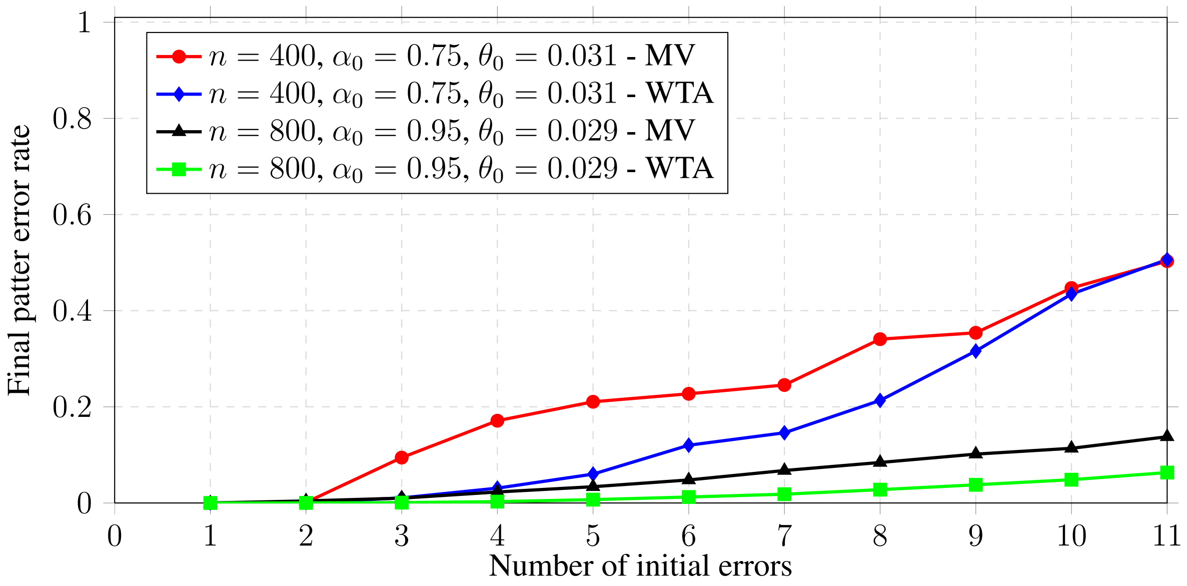


Figure 5.10: Pattern error rate against the initial number of erroneous nodes for two different network sizes $n = 800$ and $k = 400$. In both cases $k = n/2$.

to see how the suggested algorithms perform in situations where there is inherent redundancy within the patterns. Interestingly, the performance of the method in [38] deteriorates while that of [40] becomes better in certain cases (e.g., for $C = 2000$). However, neither of the approaches can achieve the small error rates obtained by the method proposed here. Thus, even though redundancy and structure exists in this particular set of patterns, the mentioned approaches could not exploit it to achieve better pattern retrieval capacities or smaller recall error rates.

5.7 Discussions

The proposed algorithms in this chapter show how the subspace assumption can substantially improve the pattern retrieval capacity. However, there are two important remarks in order. First of all, note that since we assume patterns form a subspace, any noise pattern that lies within the subspace can not even be distinguished, let alone be corrected. So how does this issue relate to the recall performance where we investigate the relationship between the final pattern error rate and the initial number of errors, irrespective of a particular noise pattern?

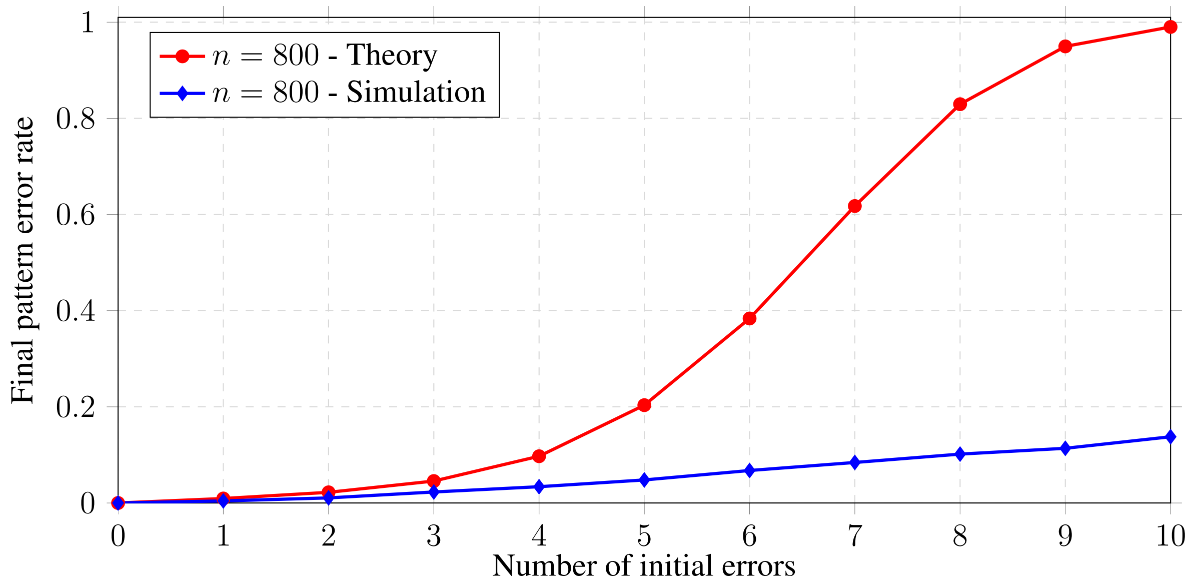


Figure 5.11: Pattern error rate against the initial number of erroneous nodes and comparison with theoretical upper bounds for $n = 800$, $k = 400$, $\alpha_0 = 0.95$ and $\theta_0 = 0.029$.

To answer this, we should first of all remember that during the recall phase, we start the algorithm from the vicinity of a given memorized pattern (local minimum), since we assume that a fairly limited number of entries is corrupted due to noise (otherwise the algorithm will not succeed). At this point, what we should address is if a noise pattern with a limited number of non-zero elements could fall into the patterns subspace. In certain cases, this could obviously happen, as depicted in Figure 5.14. Nevertheless, in other cases where the subspace is "tilted", changing a few entries in a memorized pattern will not result in a pattern that lies within the same subspace since the patterns are assumed to be *integer-valued*. Figure 5.15 illustrates this situation. In such cases, we could safely claim that we could correct any error patterns with a limited number of non-zero entries. In fact, in the following chapters, we somewhat relax this limitation by considering schemes where the neural algorithm is required to be able to deal with only a single input error during the recall phase while ensuring that the overall recall performance is not only maintained but also improved.

The second remark, which is closely related to the first one, concerns the cases where we are not interested in remembering *all* of the patterns in a given subspace, but a select number of them. Then, the question is if and how we could differentiate between a memorized

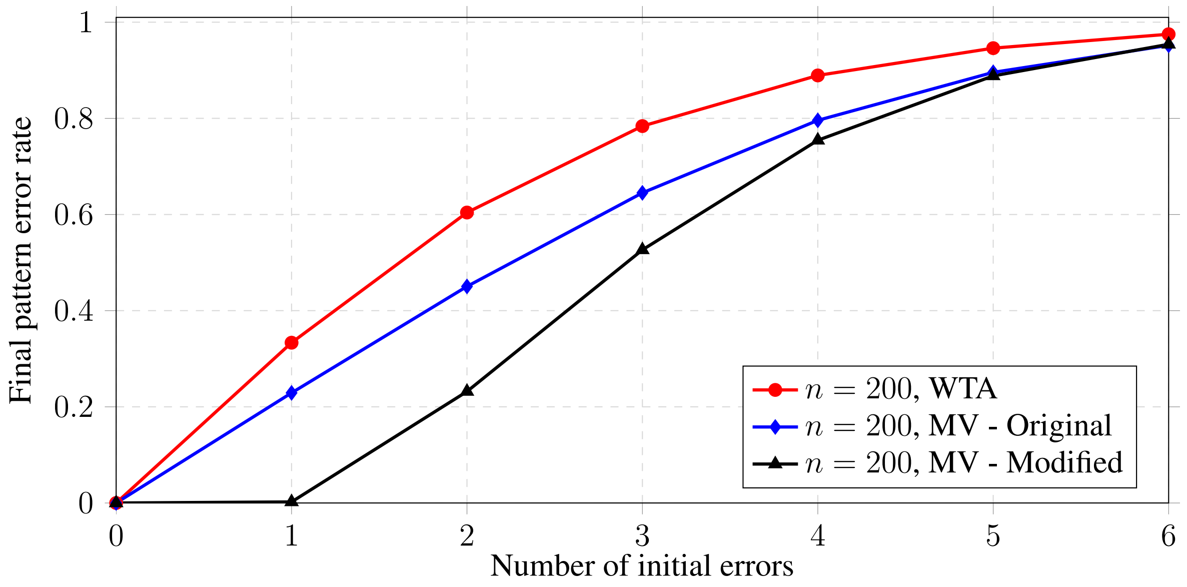


Figure 5.12: Pattern error rate against the initial number of erroneous nodes for different recall algorithms. Note how Majority-Voting (MV) algorithms performs better than the Winner-Take-All (WTA) method. Furthermore, the modified MV algorithms achieves better results than the original one.

pattern and any other pattern that lies within the same subspace during the recall phase. Once again, the key point in answering this question is to note that in the recall phase we start from the vicinity of a memorized pattern. Thus, in a tilted subspace we can retrieve the correct pattern with high probability and, hence, be able to differentiate between the set of memorized patterns and the other patterns in the same subspace.

Overall, it must be emphasized that the *theoretical* results derived in this (and subsequent) chapters show that *if* the neural connectivity matrix has certain properties, e.g., no two pattern neurons sharing the same neighborhood for the proposed algorithm in this chapter, then the recall algorithm could correct a number of erroneous symbols in the input pattern during the recall phase. Please note that these properties do not have any effect on the learning phase or the number of learned patterns, and they just ensure having a decent performance in the recall phase. In general, the closer the neural graph is to an *expander* (see Appendix 5.B), the better the recall performance will be.

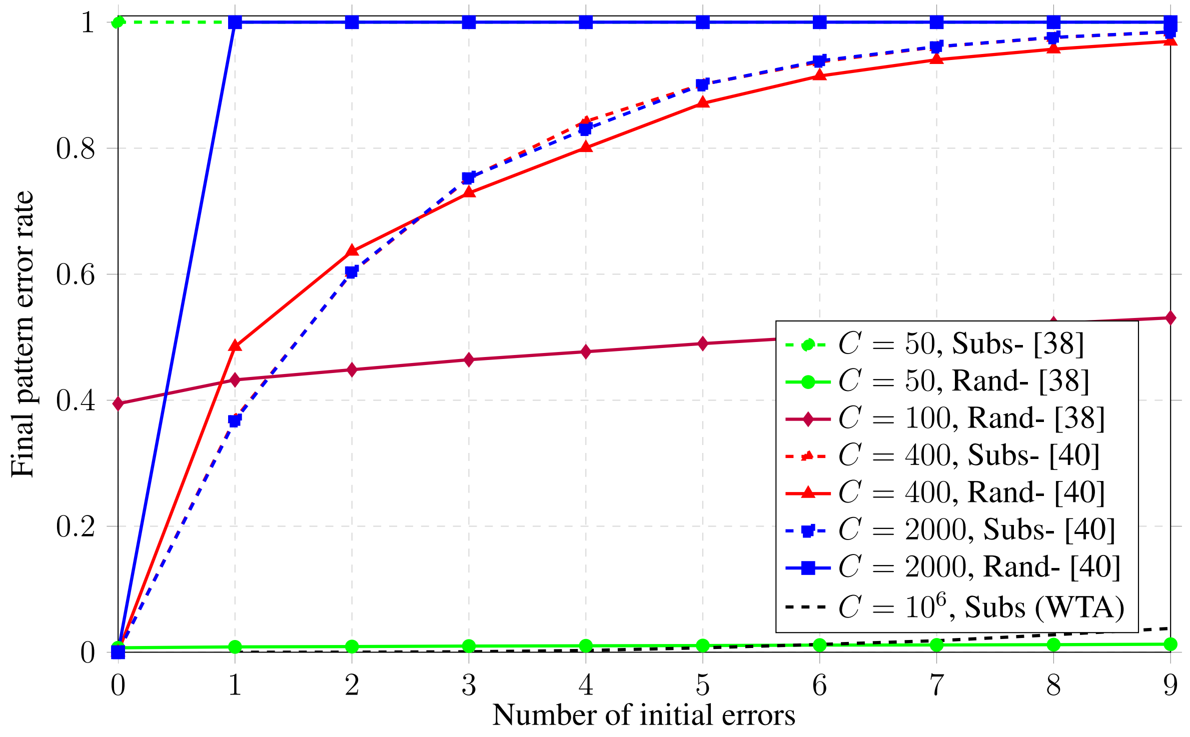


Figure 5.13: Recall error rate against the initial number of erroneous nodes for the method proposed in this chapter and those of [38] and [40]. Different pattern retrieval capacities are considered to investigate their effect on the recall error rate.

5.8 Final Remarks

In this chapter, we proposed a neural associative memory which is capable of exploiting inherent redundancy in input patterns that belong to a subspace to enjoy an exponentially large pattern retrieval capacity. Furthermore, the proposed method uses simple iterative algorithms for both learning and recall phases which makes gradual learning possible and maintains rather good recall performance. We also analytically investigated the performance of the recall algorithm by deriving an upper bound on the probability of recall error as a function of input noise. Our simulation results confirm the consistency of the theoretical results with those obtained in practice over *synthetic* datasets, for different network sizes and learning/recall parameters.

Improving the error correction capabilities of the proposed network is definitely a subject

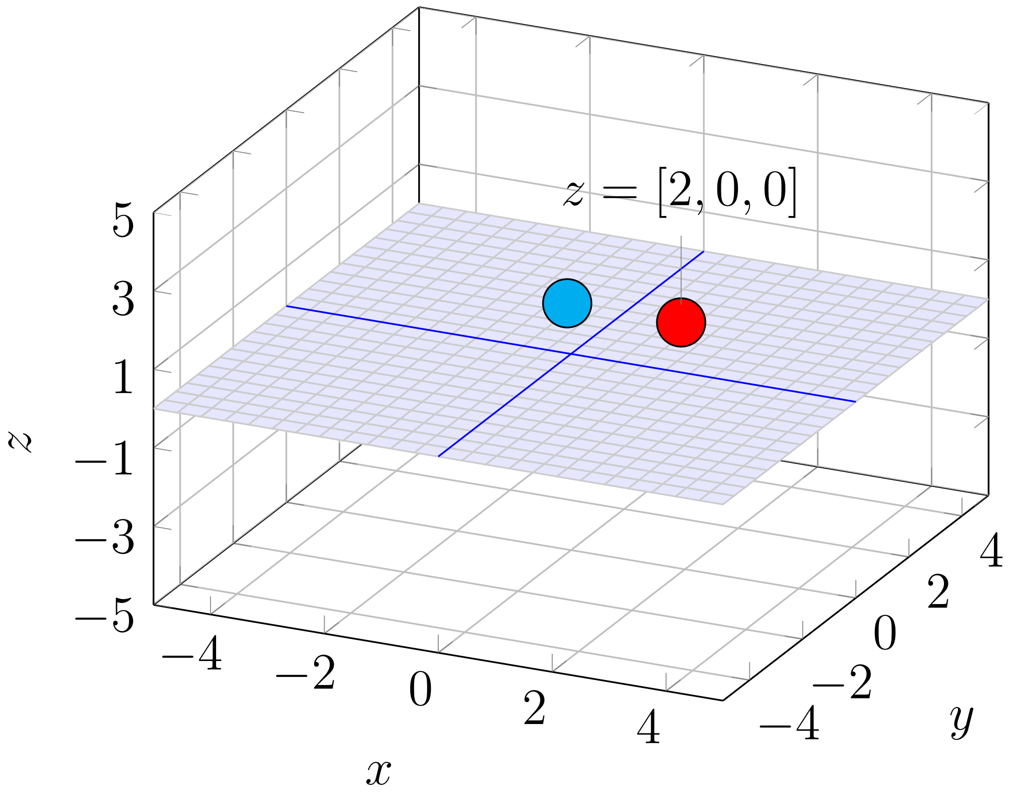


Figure 5.14: Example of a "flat" subspace where even a single error could result in a recall failure. The blue circle is the memorized pattern and the red circle represents the noisy cue. The noise vector is indicated by z.

worthy of further studies. We will address this issue in the subsequent chapters.

Extending this method to capture other sorts of redundancy, i.e., other than belonging to a subspace, will be another topic which we would like to explore in future. One special case where patterns satisfy nonlinear constraints will be considered in Chapter 9. Other interesting cases include sparse patterns which have been considered in previous art (cf. [6]). Nevertheless, further investigations in light of recent developments in the field of compressive sensing defenitely seems interesting.

Finally, practical modifications to the learning and recall algorithms is of great interest. One good example is simultaneous learn and recall capability, i.e., to have a network which learns a subset of the patterns in the subspace and moves immediately to the recall phase. During the recall phase, if the network is given a noisy version of the patterns previously memorized, it eliminates the noise using the algorithms described in this Chapter. However,

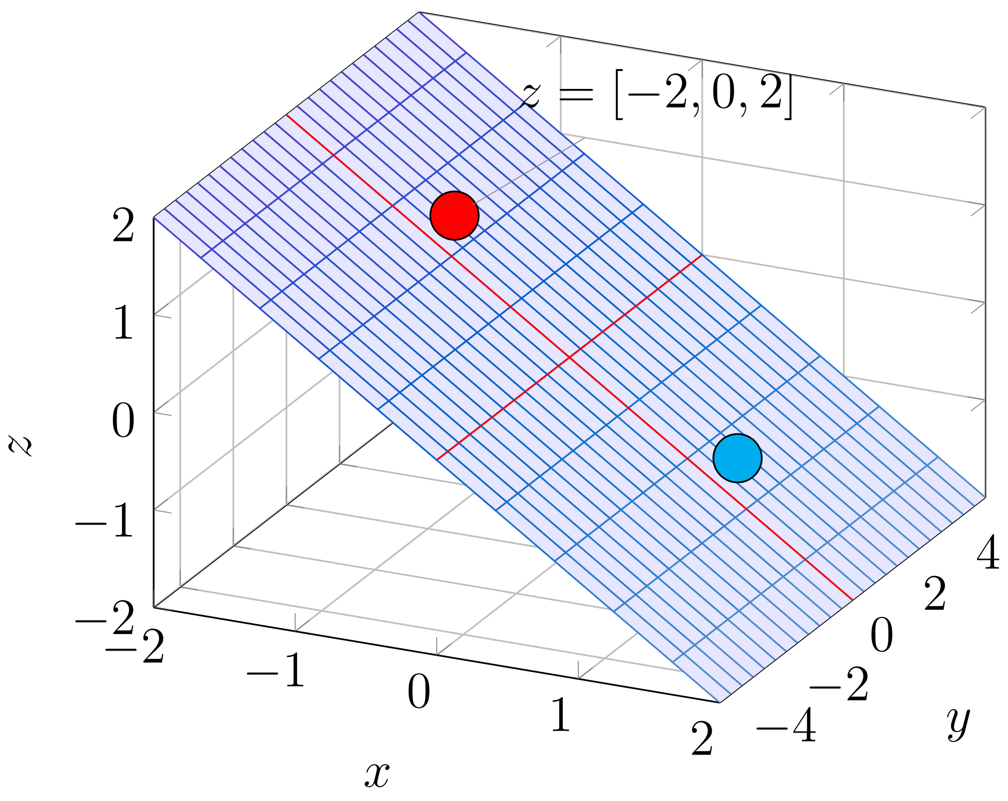


Figure 5.15: Example of a "tilted" subspace where we could correct any error pattern with a limited number of non-zero entries. The blue circle is the memorized pattern and the red circle represents the noisy cue. The noise vector is indicated by z.

if it is a new pattern, i.e., one that we have not learned yet, the network adjusts the weights in order to learn this pattern as well. Such model is of practical interest and closer to real-world neuronal networks. Therefore, it would be interesting to design a network with this capability while maintaining good error correcting capabilities and large pattern retrieval capacities.

5.A Proof of Lemma 10

In this appendix, we find an expression for the average neighborhood size for erroneous nodes, $S_t = \mathbb{E}(|\mathcal{N}(\mathcal{E}_t)|)$. Our argument is very similar to that of Motwani and Raghavan [51] to prove that random bipartite graphs are expanders. Towards this end, we assume the following procedure for constructing a right-irregular bipartite graph:

- In each iteration, we pick a variable node x with a degree randomly determined according to the given degree distribution.

- Based on the given degree d_x, we pick d_x constraint nodes uniformly at random *with replacement* and connect x to the constraint node.

- We repeat this process n times, until all variable nodes are connected.

Note that the assumption that we do the process with replacement is made to simplify the analysis. This assumption becomes more exact as n grows.

Having the above procedure in mind, we will find an expression for the average number of constraint nodes in each construction round. More specifically, we will find the average number of constraint nodes connected to i pattern nodes at round i of construction. This relationship will in turn yield the average neighborhood size of $|\mathcal{E}_t|$ erroneous nodes in iteration t of error correction algorithm described in section 5.4.

With some abuse of notation, let S_e denote the number of constraint nodes connected to pattern nodes in round e of construction procedure mentioned above. We write S_e recursively in terms of e as follows:

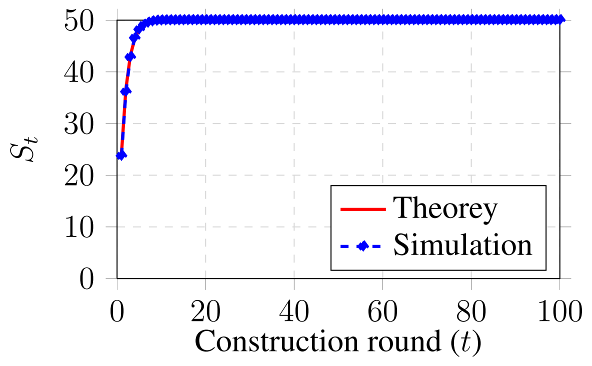
$$\begin{aligned} S_{e+1} &= \mathbb{E}_{d_x}\left\{\sum_{j=0}^{d_x}\binom{d_x}{j}\left(\frac{S_e}{m}\right)^{d_x-j}\left(1-\frac{S_e}{m}\right)^{j}(S_e+j)\right\} \\ &= \mathbb{E}_{d_x}\{S_e+d_x(1-S_e/m)\} \\ &= S_e+\bar{d}(1-S_e/m), \end{aligned} \tag{5.19}$$

where $\bar{d} = \mathbb{E}_{d_x}\{d_x\}$ is the average degree of the pattern nodes. In words, the first line calculates the average growth of the neighborhood when a new variable node is added to the graph. The following equalities directly follow from relationships of binomial sums. Noting that $S_1 = \bar{d}$, one obtains:

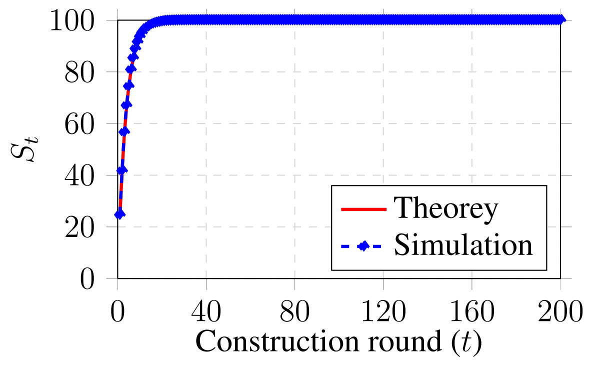
$$S_t = m\left(1-\left(1-\frac{\bar{d}}{m}\right)^{|\mathcal{E}_t|}\right). \tag{5.20}$$

This concludes the proof. □

In order to verify the correctness of the above analysis, we have performed some simulations for different network sizes and degree distributions obtained from the graphs returned by the learning algorithm. We generated 100 random graphs and calculated the average neighborhood size in each iteration over these graphs. Furthermore, two different network sizes were considered $n = 100, 200$ and $m = n/2$ in all cases, where n and m are the number of pattern and constraint nodes, respectively. The result for $n = 100, m = 50$ is shown in Figure 5.16a, where the average neighborhood size in each iteration is illustrated and compared with theoretical estimations given by equation (5.16). Figure 5.16b shows



(a) $n = 100$, $m = 50$.



(b) $n = 200$, $m = 100$.

Figure 5.16: The theoretical estimation and simulation results for the neighborhood size of irregular graphs with a given degree-distribution over 2000 random graphs for two different network sizes.

similar results for $n = 200$, $m = 100$. In the figure, the dashed line shows the average neighborhood size over these graphs. The solid line corresponds to theoretical estimations. It is obvious that the theoretical value is an exact approximation of the simulation results.

5.B Expander Graphs

This section contains the definitions and the necessary background on expander graphs.

Definition 21. *A regular (d_p, d_c, n, m) bipartite graph W is a bipartite graph between n pattern nodes of degree d_p and m constraint nodes of degree d_c.*

Definition 22. *An $(\alpha n, \beta d_p)$-expander is a (d_p, d_c, n, m) bipartite graph such that for any subset $\mathcal{P}$ of pattern nodes with $|\mathcal{P}| < \alpha n$ we have $|\mathcal{N}(\mathcal{P})| > \beta d_p |\mathcal{P}|$ where $\mathcal{N}(\mathcal{P})$ is the set of neighbors of $\mathcal{P}$ among the constraint nodes.*

The following result from [49] shows the existence of families of expander graphs with parameter values that are relevant to us.

Theorem 23. *[49] Let W be a randomly chosen $(d_p, d_c)-$regular bipartite graph between n d_p-regular vertices and $m = (d_p/d_c)$ d_c-regular vertices. Then for all $0 < \alpha < 1$, with* *high probability, all sets of αn d_p-regular vertices in W have at least*

$$n\left(\frac{d_p}{d_c}(1-(1-\alpha)^{d_c})-\sqrt{\frac{2d_c\alpha h(\alpha)}{\log_2 e}}\right)$$

neighbors, where $h(\cdot)$ is the binary entropy function.

The following result from [52] shows the existence of families of expander graphs with parameter values that are relevant to us.

Theorem 24. *Let d_c, d_p, m, n be integers, and let $\beta < 1 - 1/d_p$. There exists a small $\alpha > 0$ such that if W is a (d_p, d_c, n, m) bipartite graph chosen uniformly at random from the ensemble of such bipartite graphs, then W is an $(\alpha n, \beta d_p)$-expander with probability $1 - o(1)$, where $o(1)$ is a term going to zero as n goes to infinity.*

5.C Analysis of the Recall Algorithms for Expander Graphs

Analysis of the Winner-Take-All Algorithm

We prove the error correction capability of the winner-take-all algorithm in two steps: first we show that in each iteration, only pattern neurons that are corrupted by noise will be chosen by the winner-take-all strategy to update their state. Then, we prove that the update is in the right direction, i.e. toward removing noise from the neurons.

Lemma 25. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander, with $\beta > 1/2$, and the original number of erroneous neurons are less than or equal to 2, then in each iteration of the winner-take-all algorithm only the corrupted pattern nodes update their value and the other nodes remain intact. For $\beta = 3/4$, the algorithm will always pick the correct node if we have two or fewer erroneous nodes.*

Proof. If we have only one node x_i in error, it is obvious that the corresponding node will always be the winner of the winner-take-all algorithm unless there exists another node that has the same set of neighbors as x_i. However, this is impossible since because of the expansion properties, the neighborhood of these two nodes must have at least $2\beta d_p$ members which for $\beta > 1/2$ is strictly greater than d_p. As a result, no two nodes can have the same neighborhood and the winner will always be the correct node.

In the case where there are two erroneous nodes, say x_i and x_j, let $\mathcal{E}$ be the set $\{x_i, x_j\}$ and $\mathcal{N}(\mathcal{E})$ be the corresponding neighborhood on the constraint nodes side. Furthermore, assume x_i and x_j share $d_{p'}$ of their neighbors so that $|\mathcal{N}(\mathcal{E})| = 2d_p - d_{p'}$. Now because of the expansion properties:

$$|\mathcal{N}(\mathcal{E})| = 2d_p - d_{p'} > 2\beta d_p \Rightarrow d_{p'} < 2(1-\beta)d_p.$$

Now we have to show that there are no nodes other than x_i and x_j that can be the winner of the winner-take-all algorithm. To this end, note that only those nodes that are connected to $N(\mathcal{E})$ will receive some feedback and can hope to be the winner of the process. So let us consider such a node x_ℓ that is connected to d_{p_ℓ} of the nodes in $N(\mathcal{E})$. Let $\mathcal{E}'$ be $\mathcal{E} \cup \{x_\ell\}$ and $N(\mathcal{E}')$ be the corresponding neighborhood. Because of the expansion properties we have $|N(\mathcal{E}')| = d_p - d_{p_\ell} + |N(\mathcal{E})| > 3\beta d_p$. Thus:

$$d_{p_\ell} \;<\; d_p + |N(\mathcal{E})| - 3\beta d_p = 3d_p(1-\beta) - d_{p'}.$$

Now, note that the nodes x_i and x_j will receive some feedback from $2d_p - d_{p'}$ edges because we assume there is no noise cancellation due to the fact that neural weights are real-valued and noise entries are integers. Since $2d_p - d_{p'} > 3d_p(1-\beta) - d_{p'}$ for $\beta > 1/2$, we conclude that $d_p - d_{p'} > d_{p_\ell}$ which proves that no node outside $\mathcal{E}$ can be picked during the winner-take-all algorithm as long as $|\mathcal{E}| \leq 2$ for $\beta > 1/2$. $\square$

In the next lemma, we show that the state of erroneous neurons is updated in the direction of reducing the noise.

Lemma 26. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander, with $\beta > 3/4$, and the original number of erroneous neurons is less than or equal $e_{\min} = 2$, then in each iteration of the winner-take-all algorithm the winner is updated toward reducing the noise.*

Proof. When there is only one erroneous node, it is obvious that all its neighbors agree on the direction of update and the node reduces the amount of noise by one unit.

If there are two nodes x_i and x_j in error, since the number of their shared neighbors is less than $2(1-\beta)d_p$ (as we proved in the last lemma), then more than half of their neighbors would be unique if $\beta \geq 3/4$. These unique neighbors agree on the direction of update. Therefore, whoever the winner is will be updated to reduce the amount of noise by one unit. $\square$

The following theorem sums up the results of the previous lemmas to show that the winner-take-all algorithm is guaranteed to perform error correction.

Theorem 27. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander, with $\beta \geq 3/4$, then the winner-take-all algorithm is guaranteed to correct at least $e_{\min} = 2$ positions in error, irrespective of the magnitudes of the errors.*

Proof. The proof is immediate from Lemmas 25 and 26. $\square$

Analysis of the Majority Voting Algorithm

Roughly speaking, one would expect the Majority-Voting algorithm to be sub-optimal in comparison to the winner-take-all strategy, since the pattern neurons need to make independent decisions, and are not allowed to cooperate amongst themselves. In this subsection, we show that despite this restriction, the Majority-Voting algorithm is capable of error correction; the sub-optimality in comparison to the winner-take-all algorithm can be quantified in terms of a larger expansion factor β being required for the graph.

Theorem 28. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander with $\beta > \frac{4}{5}$, then the Majority-Voting algorithm with $\varphi = \frac{3}{5}$ is guaranteed to correct at least two positions in error, irrespective of the magnitudes of the errors.*

Proof. As in the proof for the winner-take-all case, we will show our result in two steps: first, by showing that for a suitable choice of the Majority-Voting threshold φ, that only the positions in error are updated in each iteration, and that this update is towards reducing the effect of the noise.

Case 1 First consider the case that only one pattern node x_i is in error. Let x_j be any other pattern node, for some $j \neq i$. Let x_i and x_j have $d_{p'}$ neighbors in common. As argued in the proof of Lemma 25, we have that

$$d_{p'} < 2d_p(1-\beta). \tag{5.21}$$

Hence for $\beta = \frac{4}{5}$, x_i receives non-zero feedback from at least $\frac{3}{5}d_p$ constraint nodes, while x_j receives non-zero feedback from at most $\frac{2}{5}d_p$ constraint nodes. In this case, it is clear that setting $\varphi = \frac{3}{5}$ will guarantee that only the node in error will be updated, and that the direction of this update is towards reducing the noise.

Case 2 Now suppose that two distinct nodes x_i and x_j are in error. Let $\mathcal{E} = \{x_i, x_j\}$, and let x_i and x_j share $d_{p'}$ common neighbors. If the noise corrupting these two pattern nodes, denoted by z_i and z_j, are such that $\text{sign}(z_i) = \text{sign}(z_j)$, then both x_i and x_j receive $-\text{sign}(z_i)$ along all d_p edges that they are connected to during the backward iteration. Now suppose that $\text{sign}(z_i) \neq \text{sign}(z_j)$. Then x_i (x_j) receives correct feedback from at least the $d_p - d_{p'}$ edges in $\mathcal{N}(\{x_i\})\backslash\mathcal{E}$ (resp. $\mathcal{N}(\{x_j\})\backslash\mathcal{E}$) during the backward iteration. Therefore, if $d_{p'} < d_p/2$, the direction of update would be also correct and the feedback will reduce noise during the update. In addition, from equation (5.21) we know that for $\beta = 4/5$, $d_{p'} \leq 2d_p/5 < d_p/2$. Therefore, the two noisy nodes will be updated towards the correct direction.

Let us now examine what happens to a node x_ℓ that is different from the two erroneous nodes x_i, x_j. Suppose that x_ℓ is connected to d_{p_ℓ} nodes in $\mathcal{N}(\mathcal{E})$. From the proof of Lemma 25, we know that

$$\begin{aligned} d_{p_\ell} &< 3d_p(1-\beta) - d_{p'} \\ &\leq 3d_p(1-\beta). \end{aligned}$$

Hence x_ℓ receives at most $3d_p(1-\beta)$ non-zero messages during the backward iteration.

For $\beta > \frac{4}{5}$, we have that $d_p - 2d_p(1-\beta) > 3d_p(1-\beta)$. Hence by setting $\beta = \frac{4}{5}$ and $\varphi = [d_p - 2d_p(1-\beta)]/d_p = \frac{3}{5}$, it is clear from the above discussions that we have ensured the following in the case of two erroneous pattern nodes:

- The noisy pattern nodes are updated towards the direction of reducing noise.
- No pattern node other than the erroneous pattern nodes is updated.

□

Minimum Distance of Patterns

Next, we present a sufficient condition such that the minimum Hamming distance[13] between these exponential number of patterns is not too small. In order to prove such a result, we will exploit the expansion properties of the bipartite graph W; our sufficient condition will be in terms of a lower bound on the parameters of the expander graph.

Theorem 29. *Let W be a $(d_p, d_c, n, m)-$regular bipartite graph, that is an $(\alpha n, \beta d_p)$ expander. Let $\mathcal{X}$ be the set of patterns corresponding to the expander weight matrix W. If*

$$\beta > \frac{1}{2} + \frac{1}{4d_p},$$

then the minimum distance between the patterns is at least $\lfloor \alpha n \rfloor + 1$.

Proof. Let d be less than αn, and let W_i denote the i^{th} column of W. If two patterns are at Hamming distance d from each other, then there exist non-zero integers $c_1, c_2, \dots, c_d$ such that

$$c_1 W_{i_1} + c_2 W_{i_2} + \cdots + c_d W_{i_d} = 0, \tag{5.22}$$

where $i_1, \dots, i_d$ are distinct integers between 1 and n. Let $\mathcal{P}$ denote any set of pattern nodes of the graph represented by W, with $|\mathcal{P}| = d$. As in [50], we divide $\mathcal{N}(\mathcal{P})$ into two disjoint sets: $\mathcal{N}_{unique}(\mathcal{P})$ is the set of nodes in $\mathcal{N}(\mathcal{P})$ that are connected to only one edge emanating from $\mathcal{P}$, and $\mathcal{N}_{shared}(\mathcal{P})$ comprises the remaining nodes of $\mathcal{N}(\mathcal{P})$ that are connected to more

[13]Two (possibly non-binary) vectors x and y are said to be at a Hamming distance d from each other if they are coordinate-wise equal to each other on all but d coordinates.

than one edge emanating from $\mathcal{P}$. If we show that $|\mathcal{N}_{unique}(\mathcal{P})| > 0$ for all $\mathcal{P}$ with $|\mathcal{P}| = d$, then (5.22) cannot hold, allowing us to conclude that no two patterns with distance d exist. Using the arguments in [50, Lemma 1], we obtain that

$$|\mathcal{N}_{unique}(\mathcal{P})| > 2d_p|\mathcal{P}|\left(\beta - \frac{1}{2}\right).$$

Hence no two patterns with distance d exist if

$$2d_pd\left(\beta - \frac{1}{2}\right) > 1 \Leftrightarrow \beta > \frac{1}{2} + \frac{1}{2d_pd}.$$

By choosing $\beta > \frac{1}{2} + \frac{1}{4d_p}$, we can hence ensure that the minimum distance between patterns is at least $\lfloor \alpha n \rfloor + 1$. □

Choice of Parameters

In order to put together the results of the previous two subsections and obtain a neural associative scheme that stores an exponential number of patterns and is capable of error correction, we need to carefully choose the various relevant parameters. We summarize some design principles below.

- From Theorems 24 and 29, the choice of β depends on d_p, according to $\frac{1}{2} + \frac{1}{4d_p} < \beta < 1 - \frac{1}{d_p}$.
- Choose d_c, Q, υ, γ so that Theorem 37 yields an exponential number of patterns.
- For a fixed α, n has to be chosen large enough so that an $(\alpha n, \beta d_p)$ expander exists according to Theorem 24, with $\beta \geq 3/4$ and so that $\alpha n/2 \geq e_{\min} = 2$.

Once we choose a judicious set of parameters according to the above requirements, we have a neural associative memory that is guaranteed to recall an exponential number of patterns even if the input is corrupted by errors in two coordinates.

Chapter 6

Multi-Level Neural Associative Memories

The idea introduced in the previous chapter, i.e., to memorize patterns that satisfy some linear constraints (belong to a subspace), certainly increased the pattern retrieval capacity from polynomial to exponential in terms of network size.

However, a closer look at Figure 5.8 shows that the noise elimination capabilities of the proposed algorithm is fairly limited. For instance in Figure 5.8, if 10 out of 400 pattern neurons are corrupted by noise, the recall error rate will be more than 40%. More specifically, the number of errors such a network can correct is usually a constant and does not scale with n.

In this chapter (and the next one), we consider one way of working around this problem by modifying the network architecture. More specifically, we divide the network into several blocks, each of which will be a smaller example of the network we had in the previous section. As such, each block is trained according to Algorithm 1 and the recall algorithms would be the same as Algorithms 2 or 3 (for Winner Take All and Majority Voting methods, respectively). However, now that we have several blocks that can correct at least one or two errors each, we can achieve much better performance in the recall phase by combining them together.

The architecture that is proposed in this chapter is based on dividing the pattern neurons into *non-overlapping* clusters (blocks) and then adding a second layer to take into account the inter-cluster correlations. Evidently, this approach achieves better recall error rates than the original approach proposed in Chapter 5. However, based on the architecture, it is obvious that the amount of errors that can be corrected would still be a constant with respect to n (this problem will be solved in the next chapter, when we introduce a simple but effective trick in the architecture that makes all the difference). Thus, and to summarize, in this chapter we propose a novel architecture to improve the performance of the recall phase.

Before getting into details, there is one point to mention: while dividing the network into smaller blocks help the error correction performance, it imposes a more restrictive assumptions on the patterns, namely, their sub-patterns should form a subspace. In certain circumstances, it is possible to just assume the patterns are coming from a subspace and apply the learning method proposed in the previous chapter and then, once the network is available, divide it into smaller overlapping clusters. Nevertheless, we are aware of the fact in general, this assumption is restrictive and later in the thesis, we will discuss ways to work around this problem by focusing on networks that learn non-linear constraints within the patterns.

6.1 Related Work

Modification of neural architecture has previously been used to improve the capacity. One particular work that is closely relevant to our work is that of Venkatesh [53], where the capacity is increased to $\Theta\left(b^{n/b}\right)$ for *semi-random* patterns, where $b = \omega(\ln n)$ is the size of clusters. This significant boost to the capacity is achieved by dividing the neural network into smaller fully interconnected *disjoint* blocks. This is a huge improvement but comes at the price of limited *worst-case* noise tolerance capabilities, as compared to Hopfield networks [21]. More specifically, since the network is a set of disjoint Hopfield networks of size b, the amount of error each block can correct is in the order of ϵb, for some constant $\epsilon > 0$. As a result, if the errors in the whole network are spread uniformly among the blocks, one could correct about the order of $\epsilon bn/b = \epsilon n$ errors, which is the same as the error correction capabilities of the Hopfield network. However, in case of non-uniform distribution of errors over the blocks, it can happen that there are more than ϵb errors in one block, which the block can not handle. This will lead to a retrieval error in the network. As a result, due to the non-overlapping structure of the blocks, the error correction is limited by the performance of individual blocks.

The model proposed in [53] is very similar to the one we will discuss later in this chapter, i.e., both models use disjoint blocks. However, while our focus is on sparse blocks, Venkatesh considers fully interconnected blocks. Furthermore, we utilize the smaller blocks to improve the noise tolerance capabilities of the network whereas that of [53] uses this property to improve the pattern retrieval capacity.

The worst-case error correction capabilities of both networks are limited by the performance of each individual block (although it must be mentioned that the recall performance achieved by [53] is potentially better).

The content of this chapter is joint work with Amin Karbasi. It was published in [48].

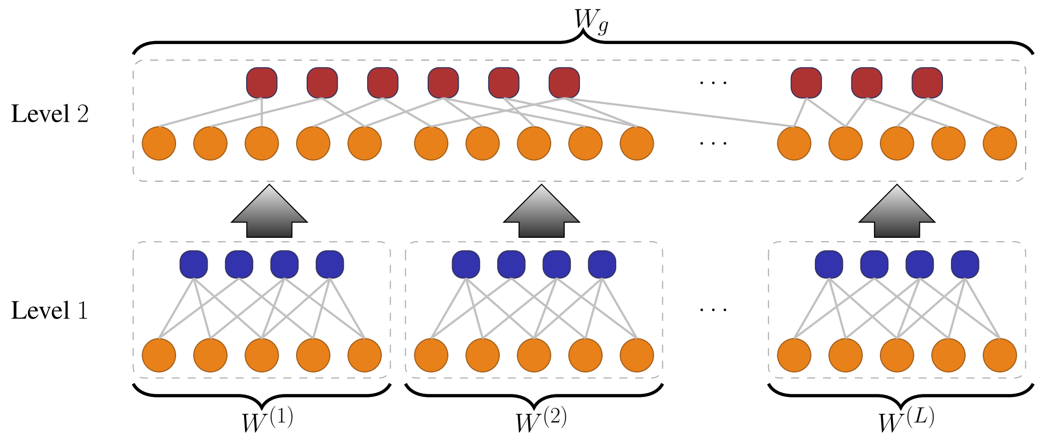


Figure 6.1: A two-level error correcting neural associative memory. Circles and squares representing pattern and constraint neurons, respectively.

Finally, the pattern retrieval capacity of the model proposed in this chapter is an exponential number in n, while the one proposed in [53] is on the order of $\ln(n)^{n/\ln(n)}$. The latter model can memorize any set of semi-random patterns, i.e. patterns that are combinations of smaller random blocks, while the former is more suitable for structured patterns that form a subspace.

6.2 Proposed Network Architecture

Figure 6.1 illustrates the proposed architecture. The network comprises two levels: the first level is divided into non-overlapping clusters. Each cluster is a bipartite graph, with circles and squares representing pattern and constraint neurons, respectively. Following the approach we proposed in the previous chapter, the connectivity matrix for each cluster is orthogonal to the *sub-patterns* that fall within its domains. The second level is virtually the same, except for its domain which contains all pattern neurons. Hence, the connectivity matrix of the second level is orthogonal to the whole patterns.

The learning algorithm for the proposed network is also the same as the algorithm we proposed in the previous chapter (see Algorithm 1). However, here the algorithm is applied to the sub-patterns that lie within each cluster. In general, the number of learned constraints in each cluster is not required to be the same.

Algorithm 5 Error Correction

```
Input: pattern matrix $\mathcal{X}$, connectivity matrix $W$, threshold $\varphi$, iteration $t_{\max}$
Output: $x_1, x_2, \dots, x_n$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the weighted input sum $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each
       neuron $y_i$ and set:
       $$y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise} \end{cases}.$$
3:     Backward iteration: Each neuron $x_j$ computes
       $$g_j = \frac{\sum_{i=1}^{m} W_{ij} y_i}{\sum_{i=1}^{m} |W_{ij}|}.$$
4:     Update the state of each pattern neuron $j$ according to $x_j = x_j - \text{sign}(g_j)$ only if
       $|g_j| > \varphi$.
5:     $t \leftarrow t + 1$
6: end for
```

6.3 Recall Phase

The recall algorithm is also similar to the one proposed in the previous chapter. Nevertheless, this time it is applied to the clusters independently, i.e., Algorithm 5, which we discussed in the previous chapter and copied here for the sake of completeness, is applied to all the clusters in the first layer. Once finished, the same algorithm is applied to the second level whose input is now the original corrupted pattern with some entries being corrected as a result of the recall algorithm in the first level.

More specifically, let L be the total number of clusters and $W^{(i)}$ be the connectivity matrix obtained from the learning algorithm for cluster i ($i = 1, \dots, L$) and W_g be the connectivity matrix of the second "global" level. Then, Algorithm 6 gives the recall algorithm for the proposed model. In words, the proposed error correction algorithm first acts upon the clusters one by one. Each time, it uses Algorithm 5 to eliminate the external errors. Then, if there were any errors left after the algorithm is done with the clusters in the first level, Algorithm 5 is applied to the second level to deal with the remaining errors.

The advantage of the proposed architecture to the one discussed in the previous chapter is obvious: if each cluster is capable of correcting e errors, the overall network could correct more than e errors on average and if we are very lucky, up to Le errors. In fact, Corollary15

Algorithm 6 Error Correction for the Multi-Level Architecture

```
Input: pattern matrix $\mathcal{X}$, Connectivity matrices $W_g$ and $W^{(1)}, \dots, W^{(L)}$
Output: $x_1, x_2, \dots, x_n$
1: for $\ell = 1 \to L$ do
2:     Apply Algorithm 5 to cluster $\ell$ with connectivity matrix $W^{(\ell)}$.
3: end for
4: if Any errors remain then
5:     Apply Algorithm 5 to $W_g$.
6: end if
```

shows that each cluster can correct one input error with high probability. However, from the simulation results of the previous section (see for instance Fig. 5.10), we know that two or more input errors usually result in high error rates. Thus, by dividing the network into smaller modules, and thus separating the error instances, we can hope for better error rates. Furthermore, once we have corrected as many errors as possible in the first level, the second level comes into play and utilizes the correlation between the clusters to correct even more errors.

6.4 Pattern Retrieval Capacity

The following theorem will prove that the proposed neural architecture is capable of memorizing an exponential number of patterns. The proof is based on construction and is given in Appendix 6.A.

Theorem 30. *Let $a > 1$ be a real number, and $Q > 2$ be an integer. Let n be an integer and $k_g = \lfloor rn \rfloor$ for some $0 < r < 1$. Then, there exists a set of $C = a^{\lfloor rn \rfloor}$ vectors in $\{0, 1, \dots, Q-1\}^n$ such that the dimension of the vector space spanned by these vectors is k_g. This dataset can be memorized by the neural network given in Figure 6.1.*

6.5 Simulation Results

Since the learning algorithm for this model is virtually the same as the one we introduced in the previous chapter, we go directly to investigating the performance of the recall phase. We have simulated the proposed algorithm in the two-level architecture with 4 equally sized clusters in the first level.[1]

[1]The MATLAB code that is used in conducting the simulations mentioned in this chapter is available online at `https://github.com/saloot/NeuralAssociativeMemory`.

Simulation scenario

We generated a synthetic dataset of $C = 10000$ patterns of length n where each block of $n/4$ belonged to a subspace of dimension $k < n/4$.[2] The result of the learning algorithm consists of four different local connectivity matrices $W^{(1)}, \dots, W^{(4)}$ as well as a global weight matrix W_g. The number of constraints in each cluster is $m = n/4 - k$ and the number of global constraints is $m_g = n - k_g$, where k_g is the dimension of the subspace for overall pattern. The learning algorithm stops when 99% of the patterns in the training set are learned.

For the recall phase, in each round we pick a pattern randomly from the training set, corrupt a given number of its symbols with ± 1 noise and use Algorithm 6 to correct the errors. As mentioned earlier, the errors are corrected first at the local and then at the global level. When finished, we compare the output of the first and the second level with the original (uncorrupted) pattern x. A pattern error is declared if the output does not match at each stage. Table 6.1 shows the simulation parameters in the recall phase.

Table 6.1: Simulation parameters

Parameter	φ	$t_{\max}$	ε
Value	0.8	$20\|z\|_0$	0.01

Results

Figure 6.2 illustrates the pattern error rates for a network of size $n = 400$ with two different values of $k_g = 100$ and $k_g = 200$. The results are also compared with that of the Majority-Voting (MV) algorithm in Chapter 5 to show the improved performance of the proposed algorithm. As one can see, having a larger number of constraints at the global level, i.e., having a smaller k_g, will result in better pattern error rates at the end of the second stage. Furthermore, note that since we stop the learning after 99% of the patterns have been learned, it is natural to see some recall errors even for 1 initial erroneous node.

Table 6.2 shows the gain we obtain by adding an additional second level to the network architecture. The gain is calculated as the ratio between the pattern error rate at the output of the first level and the pattern error rate at the output of the second level.

[2]Note that C can be an exponential number in n. However, we selected $C = 10000$ as an example to show the performance of the algorithm because even for small values of k an exponential number in k will become too large to handle numerically.

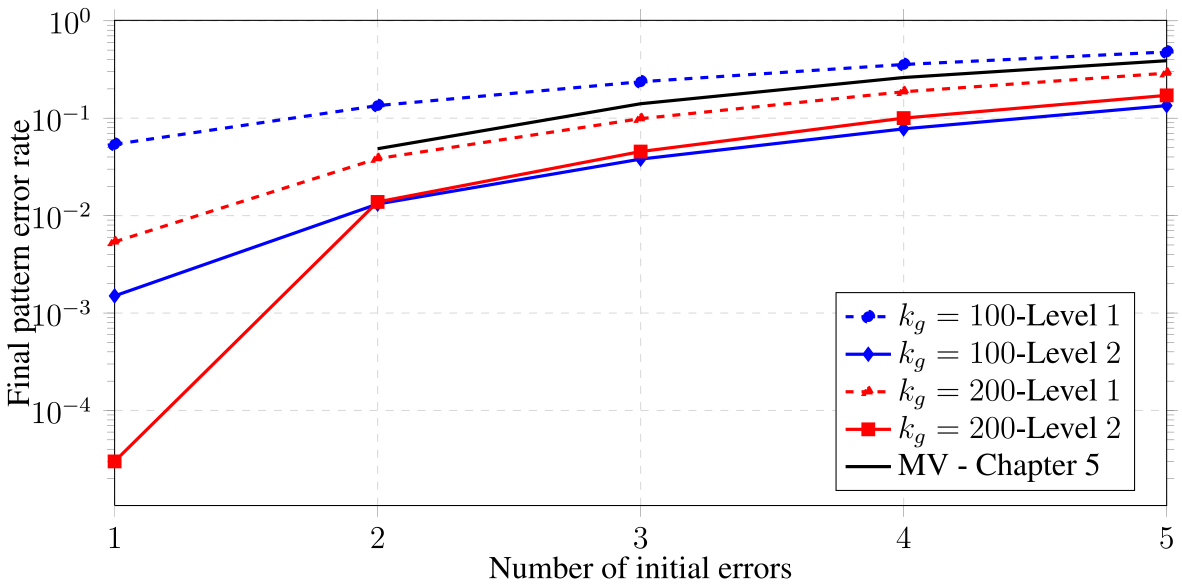


Figure 6.2: Pattern error rate against the initial number of erroneous nodes for a network of size $n = 400$ and $L = 4$ clusters in the first level. The results are compared to the Majority -Voting (MV) algorithm of Chapter 5 to highlight the improvement in the performance.

Table 6.2: Gain in Pattern Error Rate (PER) for different numbers of initial error in a network of size $n = 400$.

Number of initial errors	Gain for $k_g = 100$	Gain for $k_g = 200$
2	10.2	2.79
3	6.22	2.17
4	4.58	1.88
5	3.55	1.68

6.6 Final Remarks

In this chapter we proposed a new architecture to improve the performance of the recall phase for a neural associative memory that learns patterns belonging to a subspace. Nevertheless, there is still room to improve the performance as the number of errors that can be corrected in the recall phase is still a constant (with respect to the size of the patterns, n).

In addition, one could consider further developments of the proposed idea in different directions. As a case in point, in the two-level model discussed in this chapter we assumed that the second level enforces constraints in the same "space" as the first one, i.e., the states of the pattern neurons in the second level was the same as those of the first level. However, it is possible that the second level imposes a set of constraints in a totally different space. A good example is the case of written language. The first level could deal with local constraints on the spelling of the words, while the second level could learn more global constraints enforced by the grammar or the overall meaning of a sentence. The latter constraints are not on the space of letters but rather the space of grammar or meaning.

We could combine information from both spaces to achieve better performance in the recall phase. For instance, in order to correct an error in the word $\_at$, we can replace _ with either h, to get *hat*, or c to get *cat*. Without any other clue, we can not find the correct answer. However, if we have the sentence "*The _at ran away*", then from the constraints in the space of meanings we know that the subject must be an animal or a person. Therefore, we can return *cat* as the correct answer.

Finding a proper mapping between the two spaces that results in the best overall recall performance is an interesting topic which is unfortunately out of the scope of this thesis but could be a subject of future work.

6.A Proof of Theorem 30

The proof is very similar to the one we had in Chapter 5 and is based on construction: we construct a data set $\mathcal{X}$ with the required properties, namely the entries of patterns should be non-negative, patterns should belong to a subspace of dimension $k_g < n$ and each sub-pattern of length n/L belongs to a subspace of dimension $k < n/L$.

To start, consider a matrix $G \in \mathbb{R}^{k_g \times n}$ with rank k_g and $k_g = \lfloor rn \rfloor$, with $0 < r < 1$. Let the entries of G be non-negative integers, between 0 and $\gamma - 1$, with $\gamma \geq 2$. Furthermore, let $G_1, \dots, G_L$ be the L sub-matrices of G, where G_i consists of the columns $1 + (i-1)L$ to iL of G. Finally, assume that *in each sub-matrix* we have exactly k non-zero rows with $k < n/L$.

We start by constructing the patterns in the data set as follows: consider a random vector $u^\mu \in \mathbb{R}^{k_g}$ with integer-valued-entries between 0 and $\upsilon - 1$, where $\upsilon \geq 2$. We set the pattern $x^\mu \in \mathcal{X}$ to be $x^\mu = u^\mu \cdot G$, *if* all the entries of x^μ are between 0 and $Q - 1$. Obviously, since both u^μ and G have only non-negative entries, all entries in x^μ are non-negative. Therefore, it is the $Q - 1$ upper bound that we have to worry about.

The j^{th} entry in x^μ is equal to $x_j^\mu = \langle u^\mu, g_j \rangle$, where g_j is the j^{th} column of G. Suppose g_j has d_j non-zero elements. Then, we have:

$$x_j^\mu = \langle u^\mu, g_j \rangle \leq d_j(\gamma - 1)(\upsilon - 1)$$

Therefore, denoting $d^* = \max_j d_j$, we could choose γ, υ and d^* such that

$$Q - 1 \geq d^*(\gamma - 1)(\upsilon - 1) \tag{6.1}$$

to ensure all entries of x^μ are less than Q.

As a result, since there are υ^{k_g} vectors u with integer entries between 0 and $\upsilon - 1$, we will have $\upsilon^{k_g} = \upsilon^{\lfloor rn \rfloor}$ patterns forming $\mathcal{X}$. Which means $C = \upsilon^{\lfloor rn \rfloor}$, which would be an exponential number in n if $\upsilon \geq 2$.

Chapter 7

Convolutional Neural Associative Memories

In Chapter 5 we introduced an idea to increase the pattern retrieval capacity significantly. However, as the noise tolerance capability of the proposed method was gravely limited, in Chapter 6 we modified the proposed neural architecture to not only have exponentially large pattern retrieval capacities, but also improved error correction and noise tolerance. Nevertheless, the error correction ability of the network is still limited as it only can correct a constant number of external errors (with respect to the length of the patterns, n).

In this chapter, we build upon the architecture proposed in the previous chapter to achieve linear error correction capabilities, i.e., being able to correct a linear number of input errors (in terms of n). The new architecture is based on the same idea of breaking down the network into smaller blocks. However, instead of having disjoint blocks, this time the blocks will have some overlap with each other. Although having smaller blocks means a stricter set of constraints on the input data, this simple trick will make the difference in the noise tolerance properties of the network. We show that the proposed approach could still memorize an exponential number of patterns (with sub-patterns belonging to a subspace) and provide theoretical estimations of the recall performance.

Interestingly, the proposed model and the recall algorithm is very similar to codes on graphs [54] and the Peeling Decoder used in communication systems to deal with erasure noise [55]. As a result, we adopt methods from coding theory to theoretically analyze the behavior of the network in dealing with input errors during the recall phase. Later in the chapter we verify the theoretical analysis by numerical simulations as usual.

7.1 Related Work

With regard to dividing the input pattern to a number of overlapping smaller patches, our model is extremely similar to the convolutional neural networks and convolutional Deep Belief Networks in particular. Deep Belief Networks (DBNs) are typically used to extract/classify features by means of several consecutive stages (e.g., pooling, rectification, etc). Having multiple stages helps the network to learn more interesting and complex features. An important class of DBNs are convolutional DBNs where the input layer (or the receptive field) is divided into multiple possibly overlapping patches, and the network extract features from each patch [57].

Since we divide the input patterns into a few overlapping smaller clusters, our model is similar to that of convolutional DBNs. Furthermore, since we learn (extract) multiple *features*[1] from each block and our feature extraction matrices are not identical over different patches, our model, for instance, looks very similar to that of [58].

However, in contrast to convolutional DBNs, the focus of this work is not classification but rather recognition of the *exact* patterns from their noisy versions. Moreover, in most DBNs, we not only have to find the proper basis to represent the patterns in the feature space, but also need to calculate the projection coefficients for each input pattern. As a result, the complexity of the system is increased in pattern retrieval cases. In contrast, since our model is based on learning the dual basis of the null space, there is no need to calculate the projection coefficients in order to retrieve the pattern. We just eliminate the contribution of noise and retrieve the original pattern. Furthermore, being a single layer architecture, the proposed model in this chapter will have a faster learning phase as compared to deep multi-layer structures. In addition, the overlapping nature of the clusters allows the gradual diffusion of information over the network, which is achieved in DBNs [59] by the help of having multiple layers.

Another important point that is worth mentioning is that learning a set of input patterns with robustness against noise is not just the focus of neural associative memory. For instance, [60] proposes an interesting approach to extract robust features in autoencoders. Their approach is based on artificially introducing noise during the learning phase and letting the network learn the mapping between the corrupted input and the correct version. This way, they shift the burden from the recall phase to the learning phase. As such, this approach is quite general in that it can handle any dataset without any particular restriction. We, on the other hand, consider a more particular form of redundancy and enforce a suitable structure

The content of this chapter is joint work with Amin Karbasi and Amin Shokrollahi. It was published in [56].

[1]The features here refer to the dual vectors for each patch.

which helps us design algorithms that are faster for these particular datasets and *guaranteed* to correct a linear fraction of noise, without previously being exposed to noise.

7.2 Problem Formulation and the Model

Learning phase

Each pattern $x = (x_1, x_2, \dots, x_n)$ is a vector of length n, where $x_i \in \mathcal{Q} = \{0, \dots, Q-1\}$ for $i \in [n]$ and some non-negative integer Q. In this chapter, our focus is on memorizing patterns with strong *local correlation* among the entries. More specifically, we divide the entries of each pattern x into L *overlapping sub-patterns* of lengths $n_1, \dots, n_L$, so that $\sum n_i \geq n$. Note that due to overlaps, a pattern node can be a member of multiple sub-patterns, as shown in Figure 7.1. We denote the i^{th} sub-pattern by $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_{n_i}^{(i)})$. To enforce local correlations, we assume that the sub-patterns $x^{(i)}$ form a subspace of dimension $k_i < n_i$. This is done by imposing linear constraints on each cluster. These linear constraints are captured in the form of dual vectors as follows. In the learning phase, we would like to memorize these patterns by finding a set of non-zero vectors $w_1^{(i)}, w_2^{(i)}, \dots, w_{m_i}^{(i)}$ that are orthogonal to the set of sub-patterns $x^{(i)}$, i.e.,

$$y_j^{(i)} = (w_j^{(i)})^\top \cdot x^{(i)} = 0, \quad \forall j \in [m_i]\, \forall i \in [L], \tag{7.1}$$

where $[q]$ denotes the set $\{1, 2, \cdots, q\}$.

The weight matrix $W^{(i)} = [w_1^{(i)} | w_2^{(i)} | \dots | w_{m_i}^{(i)}]^\top$ of cluster i is created by putting all the dual vectors next to each other. Equation (7.1) can be written equivalently as

$$W^{(i)} \cdot x^{(i)} = 0.$$

One can easily map the local constraints imposed by the $W^{(i)}$'s into a global constraint by introducing a global weight matrix W of size $m \times n$. The first m_1 rows of the matrix W correspond to the constraints in the first cluster, rows $m_1 + 1$ to $m_1 + m_2$ correspond to the constraints in the second cluster, and so forth. Hence, by inserting zero entries at proper positions, we can construct the global constraint matrix W. We will use both the local and global connectivity matrices to eliminate noise in the recall phase.

Recall phase

In the recall phase a noisy version, say y, of an already learned pattern $x \in \mathcal{X}$ is given. Here, we assume that the noise is an additive vector of size n, denoted by e, whose entries assume values independently from $\{-1, 0, +1\}$ with corresponding probabilities $p_{-1} = p_{+1} = p_e/2$

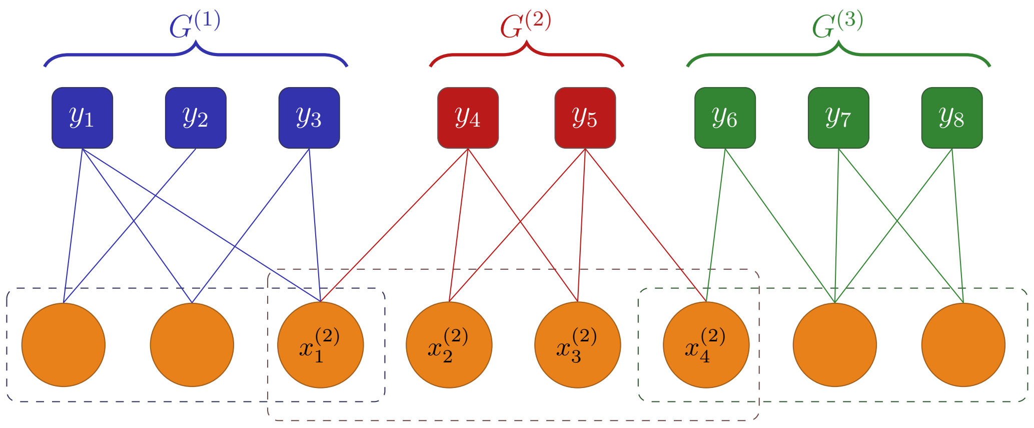


Figure 7.1: Bipartite graph G.

and $p_0 = 1 - p_e$ (i.e., each entry in the noise vector is set to ± 1 with probability p_e). We denote by $e^{(i)}$, the realization of noise on the sub-pattern $x^{(i)}$. In formula, $y = x + e$.[2] Note that $W \cdot y = W \cdot e$ and $W^{(i)} \cdot y^{(i)} = W^{(i)} \cdot e^{(i)}$. Therefore, the goal will be to remove the noise e and obtain the desired pattern x as the correct states of the pattern neurons. This task will be accomplished by exploiting the fact that we have chosen the set of patterns $\mathcal{X}$ to satisfy the set of constraints $W^{(i)} \cdot x^{(i)} = 0$.

Remark 31. *In order for the recall algorithm to work effectively, the amount of overlap between clusters should be in such a way that (almost) all pattern neurons are members of multiple clusters (more than one). The higher this number is, the better the recall algorithm's performance will be. This point will become more clear once we analyze the performance of the proposed algorithm.*

Capacity

The last issue we look at in this work is the retrieval capacity $\mathcal{C}$ of our proposed method. By construction, we show that the retrieval capacity of our network is exponential in the size of the network.

[2]Note that since entries of y should be between 0 and $Q-1$, we cap values below 0 and above $Q-1$ to 0 and $Q-1$, respectively.

Algorithm 7 Sequential Peeling Algorithm

```
Input: $\widetilde{G}, G^{(1)}, G^{(2)}, \ldots, G^{(L)}$.
Output: $x_1, x_2, \ldots, x_n$
1: while There is an unsatisfied $v^{(\ell)}$, for $\ell = 1, \ldots, L$ do
2:     for $\ell = 1 \to L$ do
3:         If $v^{(\ell)}$ is unsatisfied, apply Algorithm 5 to cluster $G^{(l)}$.
4:         If $v^{(\ell)}$ remained unsatisfied, revert the state of pattern neurons connected to $v^{(\ell)}$ to
           their initial state. Otherwise, keep their current states.
5:     end for
6: end while
7: Declare $x_1, x_2, \ldots, x_n$ if all $v^{(\ell)}$'s are satisfied. Otherwise, declare failure.
```

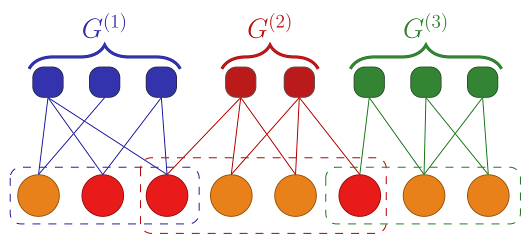
7.3 Recall Phase

Since the learning algorithm is the same as before, we directly go to discuss the details of the recall phase and assume that the connectivity matrix for each cluster i (denoted by $W^{(i)}$) has been learned, using Algorithm 1.

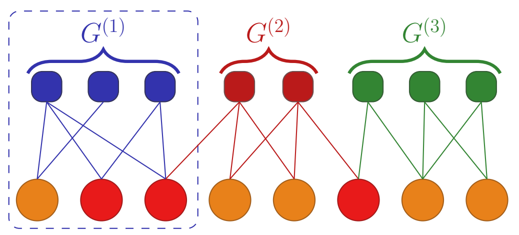
The recall phase of the proposed model consists of two parts: intra-module and inter-module. In the intra-module part, each cluster tries to remove noise from its sub-pattern by applying Algorithm 5 (explained in Chapter 6). From Corollaries 15 17 in Chapter 5 we know that using Algorithm 5, each cluster could correct at least one error with high probability, which is sufficient for our purpose in this chapter. To facilitate the analysis further down, let P_c denote this probability of correcting one error averaged over all clusters. Thus, from now on we assume that if there is a single error in a given cluster, the cluster corrects it with probability P_c and declares a failure, if there is more than one error. In case of a success, the pattern neurons keep their new states and, otherwise, revert back to their original states.

During the inter-module decoding, once the correction in cluster ℓ finishes (using Algorithm 5), Algorithm 5 is applied to cluster $\ell + 1$ and this process continues several rounds. Here, and as shown by Figure 7.2, the overlapping structure of the clusters helps them correct a linear fraction of errors together. The inter-module method is formally given in Algorithm 7.

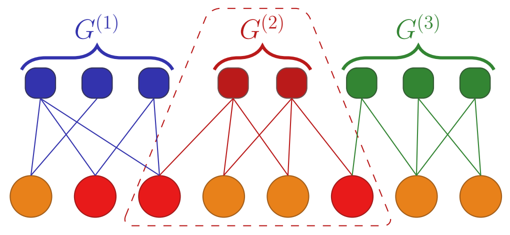
Interestingly, the inter-module algorithm is very similar to a famous decoding algorithm in communication systems for erasure channels, called the *Peeling Algorithm* [55]. To see the connections more clearly, we have to first define a "contracted" version of the neural graph in Figure 7.1. In the contracted graph $\widetilde{G}$, we compress all constraint nodes of a cluster $G^{(\ell)}$ into a single *super constraint node* $v^{(\ell)}$ (see Figure 7.3). Then, each super constraint node is capable of correcting any single error among its neighbors (in pattern neurons) or



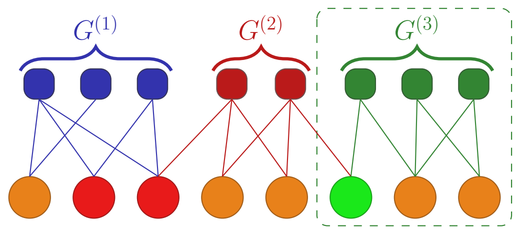
(a) Initial step



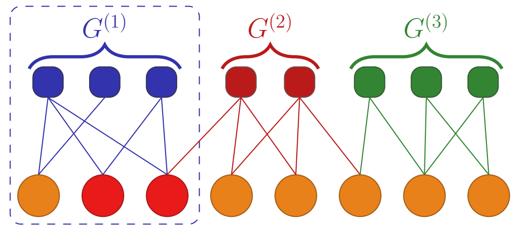
(b) Step 1: cluster 1 fails.



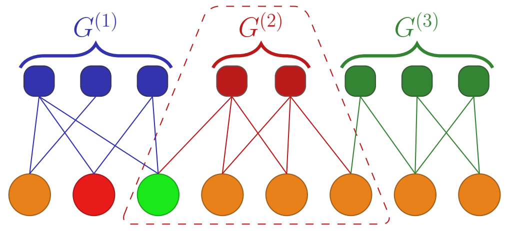
(c) Step 2: cluster 2 fails.



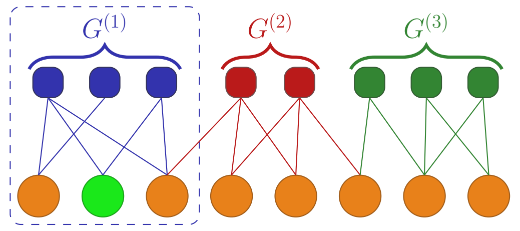
(d) Step 3: cluster 3 succeeds.



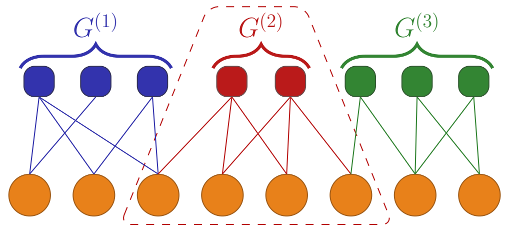
(e) Step 4: cluster 1 fails again.



(f) Step 5: cluster 2 succeeds.



(g) Step 7: cluster 1 succeeds.



(h) Step 8: Algorithm finishes successfully.

Figure 7.2: How overlap among clusters help the network to achieve better error correction. In this figure, it is assumed that each cluster could correct one input error and declares a failure if the number of input errors are higher than one.

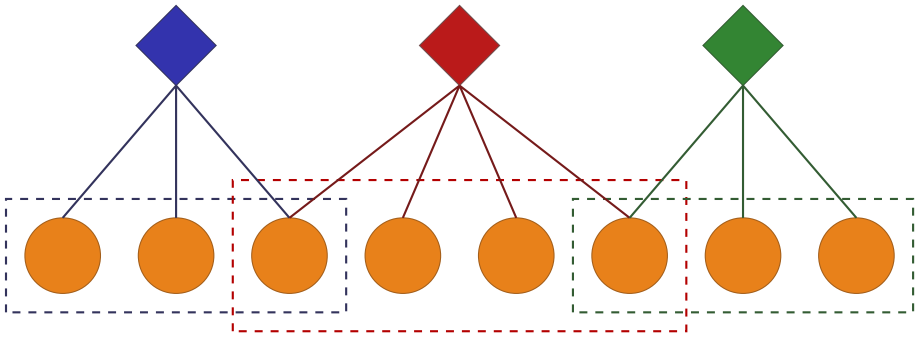


Figure 7.3: Contraction graph $\widetilde{G}$ corresponding to graph G in Figure 10.1a.

declare a failure if two or more of its neighbors are corrupted by noise. Once a single error is corrected by any cluster, this correction removes the number of errors in other clusters as well, hopefully helping them to eliminate their share of errors.

The similarity to the Peeling Decoder becomes apparent: in the Peeling Decoder, each constraint (checksum) node is capable of correcting a single erasure in its neighbors, and otherwise returns an erasure, indicating a decoding failure. Furthermore, once an erasure is eliminated by a checksum node, it helps other constraint nodes that were connected to the recently-eliminated erased node as they will have one fewer erasure among their neighbors to deal with.

Now that the similarity to graph decoding techniques has become apparent, we will use methods from modern coding theory to obtain a theoretical estimates on the error rate of the proposed recall algorithm. More specifically, we are going to use the Density Evolution (DE) method developed by Luby et al. for the erasure channels [55] and generalized by Richardson and Urbanke [54] later.

For the purpose of the analysis, let $\widetilde{\lambda}_i$ $(\widetilde{\rho}_j)$ denote the fraction of *edges* that are adjacent to pattern (constraint) nodes of degree i (j). We call $\{\widetilde{\lambda}_1, \dots, \widetilde{\lambda}_L\}$ and $\{\widetilde{\rho}_1, \dots, \widetilde{\rho}_n\}$ the pattern and super constraint degree distribution from the edge perspective, respectively. Furthermore, it is convenient to define the degree distribution polynomials as

$$\widetilde{\lambda}(z) = \sum_i \widetilde{\lambda}_i z^{i-1} \text{ and } \widetilde{\rho}(z) = \sum_i \rho_i z^{i-1}.$$

We say that the node $v^{(\ell)}$ is unsatisfied if it is connected to a noisy pattern node. Furthermore, we will need the following definition on the decision subgraph and the "tree assumption".

Definition 32. *Consider a given cluster $v^{(\ell)}$ and a pattern neuron x connected to $v^{(\ell)}$. The decision subgraph of x is a the subgraph rooted at x and branched out from the super con-*

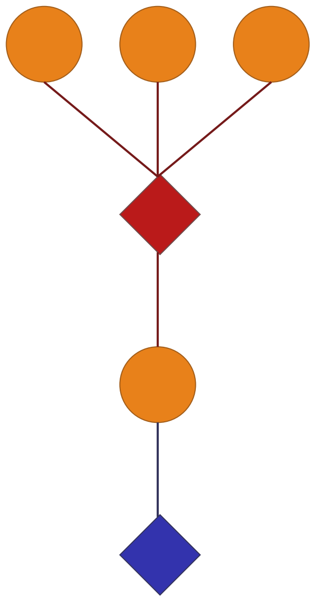


Figure 7.4: The decision subgraph of the third edge (from left) in Figure 7.3 to a depth of 2.

straint nodes, excluding $v^{(\ell)}$. If the decision subgraph is a tree up to a depth of τ, we say the tree assumption holds for τ levels.

An example of the decision subgraph for the third pattern neuron of Figure 7.3 is shown in Figure 7.4.

If the decision subgraphs for the pattern neurons in graph $\widetilde{G}$ are tree-like for a depth of τL, where τ is the total number of number of iterations performed by Algorithm 7, then the following theorem provides us with an estimate on the asymptotic performance of Algorithm 7 and the recall phase.

Theorem 33. *Let P_c be the average probability of a super constraint node correcting a single error in its domain. Then under the assumptions that graph $\widetilde{G}$ grows large and it is chosen randomly with degree distributions given by $\widetilde{\lambda}$ and $\widetilde{\rho}$, Algorithm 7 is successful if $p_e \cdot \widetilde{\lambda}\left(1 - P_c\widetilde{\rho}(1-z)\right) < z$ for $z \in (0, p_e)$.*

Proof. The proof is similar to Theorem 3.50 in [54]. Each super constraint node receives an error message from its neighboring pattern nodes with probability $z(t)$ in iteration t. Now consider a message transmitted over an edge from a given cluster $v^{(\ell)}$ to a given *noisy* pattern neuron at iteration t of Algorithm 7. This message will be a failure with probability $\pi^{(\ell)}(t)$ (indicating that the super constraint node being unable to correct the error) if

1. the super constraint node $v^{(\ell)}$ receives at least one error message from its *other* neighbors among pattern neurons, i.e., if it is connected to more than one noisy pattern neuron, or,
2. the super constraint node $v^{(\ell)}$ does not receive an error message from any of its other neighbors but is unable to correct the single error in the given noisy neuron (which occurs with probability $1 - P_c$).

Thus, we have

$$\pi^{(\ell)}(t) = 1 - P_c(1 - z(t))^{\widetilde{d}_\ell - 1} \tag{7.2}$$

As a result, if $\pi(t)$ shows the average probability that a super constraint node sends a message declaring the violation of at least one of its constraint neurons, we will have,

$$\pi(t) = \mathbb{E}_{\widetilde{d}_\ell}\{\pi^{(\ell)}(t)\} = \sum_i \widetilde{\rho}_i(1 - P_c(1 - z(t))^{\widetilde{d}_\ell - 1}) = 1 - P_c\widetilde{\rho}(1 - z(t)). \tag{7.3}$$

Now consider the message transmitted from a given pattern neuron x_i with degree d_i to a given super constraint node, $v^{(\ell)}$ in iteration $t + 1$ of Algorithm 7. This message will indicate a noisy pattern neuron if the pattern neuron was noisy in the first place and all of its *other* neighbors among super constraint nodes has sent a violation message in iteration t. Therefore, the probability of this node being noisy will be $z(0)\pi(t)^{d_i - 1}$. As a result, noting that $z(0) = p_e$, the average probability that a pattern neurons remains noisy will be

$$z(t + 1) = p_e \cdot \sum_i \widetilde{\lambda}_i \pi(t)^{i-1} = p_e \cdot \widetilde{\lambda}(\pi(t)) = p_e \cdot \widetilde{\lambda}(1 - P_c\widetilde{\rho}(1 - z(t))). \tag{7.4}$$

Therefore, the decoding operation will be successful if $z(t + 1) < z(t),\ \forall t$. As a result, we must look for the maximum p_e such that we will have $p_e \cdot \widetilde{\lambda}(1 - P_c\widetilde{\rho}(1 - z)) < z$ for $z \in [0, p_e]$. □

The condition given in Theorem 33 can be used to calculate the maximal fraction of errors Algorithm 7 can correct for the given degree distributions. For instance, for the degree distribution pair $(\widetilde{\lambda}(z) = z^2, \widetilde{\rho}(z) = z^5)$, the threshold is $p_e \approx 0.429$, below which Algorithm 7 corrects all the errors with high probability. Note that the predicted threshold by Theorem 33 is based on the assumption that a cluster can only correct a single error. In practice, as we noted earlier, a cluster can correct more. Hence, the threshold predicted by Theorem 33 is a lower bound on the overall recall performance of our neural network (as the size of the network grows).

Remark 34. *Note that when graph $\widetilde{G}$ is constructed randomly according to the given degree distribution, then as the graph size grows, the decision subgraph will become a tree with* *probability close to* 1*. Furthermore, it can be shown that the recall performance for any such graph will be* concentrated *around the average case given by Theorem 33.*

However, it must be noted that since the size of the neural graphs in this thesis are fairly limited, in some cases the decision subgraphs might not be tree-like. In those cases the performance might deviate from the one predicted by Theorem 33. Our simulation results, however, show that in many cases the proposed analytical approximation is a good estimate of the performance in the recall phase.

Overall, it must be emphasized that the theoretical results derived in this chapter show that if *the neural connectivity matrix has certain properties, e.g., no two pattern neurons sharing the same neighborhood in each cluster, then the recall algorithm could correct a linear number of erroneous symbols in the input pattern during the recall phase.*

7.4 Pattern Retrieval Capacity

Following a similar approach to Chapter 5, it is not hard to show that the proposed convolutional model has also an exponential pattern retrieval capacity. The idea is the same as before: show that there exist a dataset $\mathcal{X}$ with the desired requirement which has an exponential number of patterns.

For the sake of brevity, we postpone the formal proof to Chapter 8, where we prove a more general theorem that explicitly shows a convolutional and a coupled associative memory both have an exponential capacity (see Section 8.5 for further details).

7.5 Simulation Results

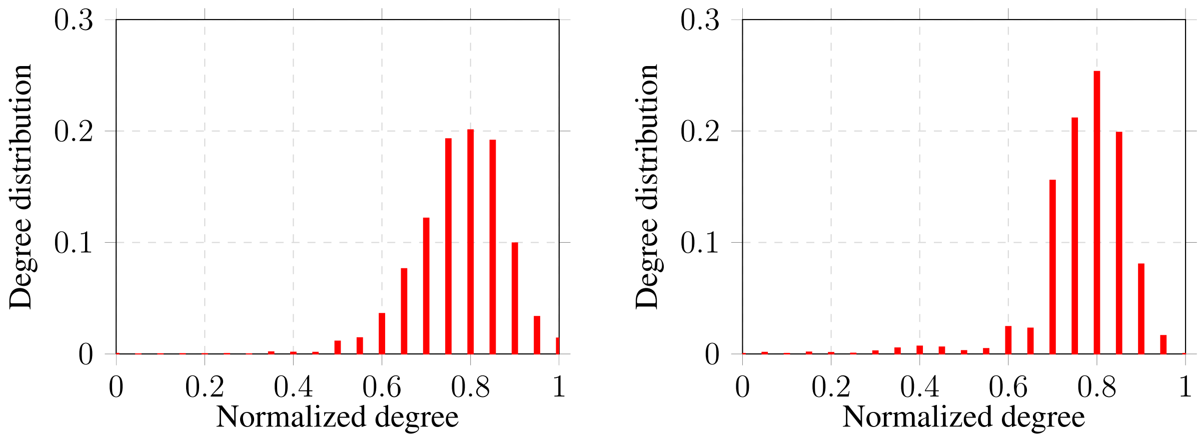
We have performed simulations over synthetic datasets to investigate the performance of the proposed algorithm and confirm the accuracy of our theoretical analysis. [3]

Simulation Scenario

There is a systematic way of generating patterns satisfying a set of linear constraints (see the proof of Theorem 37). In our simulations, we assume that each pattern neuron is connected to approximately 5 clusters. The number of connections should be neither too small (to ensure information propagation) nor too big (to adhere to the sparsity requirement).

In the learning phase, Algorithm 1 is performed in parallel for each cluster which results in the connectivity matrix for each cluster. In the recall phase, at each round, a pattern x is

[3]The MATLAB codes that are used in conducting the simulations mentioned in this chapter are available online at `https://github.com/saloot/NeuralAssociativeMemory`.



(a) Pattern neuron degrees

(b) Constraint neuron degrees

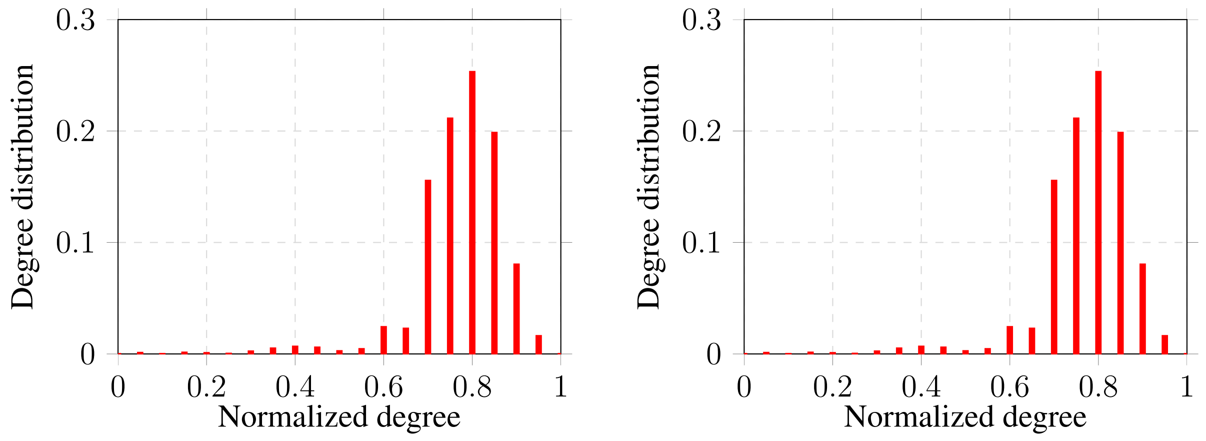
Figure 7.5: Pattern and constraint neuron degree distributions for $n = 400$, $L = 50$, and an average of 20 constraints per cluster. The learning parameters are $\alpha_t \propto 0.95/t$, $\eta = 0.75/\alpha_t$ and $\theta_t \propto 0.05/t$.

sampled uniformly at random from the training set. Then, each of its entries gets corrupted independently with probability p_e. Afterwards, Algorithm 7 is used to denoise the corrupted pattern. We repeat this process many times to calculate the error rate, and compare it to the bound derived in section 7.3.

Learning Results

The left and right panels in Figure 7.5 illustrate the degree distributions of pattern and constraint neurons, respectively, over an ensemble of 5 randomly generated networks. The network size is $n = 400$, which is divided into 50 overlapping clusters, each of size around 40, i.e., $n_\ell \simeq 40$ for $\ell = 1, \dots, 50$. Each pattern neuron is connected to 5 clusters, on average. The horizontal axis shows the normalized degree of pattern (resp., constraint) neurons and the vertical axis represents the fraction of neurons with the given normalized degree. The normalization is done with respect to the number of pattern (resp., constrain) neurons in the cluster. The parameters for the learning algorithm are $\alpha_t \propto 0.95/t$, $\eta = 0.75/\alpha_t$ and $\theta_t = 0.05$.

Figure 7.6 illustrates the same results but for a network size of $n = 960$, which is divided into 60 clusters, each with size 80, on average. The learning parameters are the same as before, i.e., $\alpha_t \propto 0.95/t$, $\eta = 0.75/\alpha_t$ and $\theta_t = 0.05$, and each pattern neuron is connected



(a) Pattern neurons degrees (b) Constraint neurons degrees

Figure 7.6: Pattern and constraint neuron degree distributions for $n = 960$, $L = 60$, and an average of 40 constraints per cluster. The learning parameters are $\alpha_t \propto 0.95/t$, $\eta = 0.75/\alpha_t$ and $\theta_t \propto 0.05/t$.

to 5 clusters on average. Note that the overall normalized degrees are smaller compared to the case of $n = 400$, which indicates sparser clusters on average.

In almost all cases, the learning phase converged within two learning iterations, i.e., by going over the data set only twice.

Recall Results

Figure 7.7 illustrates the performance of the recall algorithm. The horizontal and vertical axes represent the number of initial erroneous neurons and the final pattern error rate, respectively. The performance is compared with the theoretical bound derived in section 7.3, as well as the results of the algorithms proposed in Chapters 5 and 6. The parameters used for this simulation are $n = 400$, $L = 50$ and $\varphi = 0.82$ in the recall Algorithm 5. For the mode proposed in Chapter 6, the network size is $n = 400$ with 4 clusters in the first level and one cluster in the second level.

Note that in the theoretical estimate used in Figure 7.7, we once calculated the probability of correcting a single error by each cluster, P_c, using the lower bound in Corollary 15 in Chapter 5, and once fixed it to 1. The corresponding graphs in the figure show that the second estimation is tighter in this case and each cluster could correct a single error with probability close to 1.

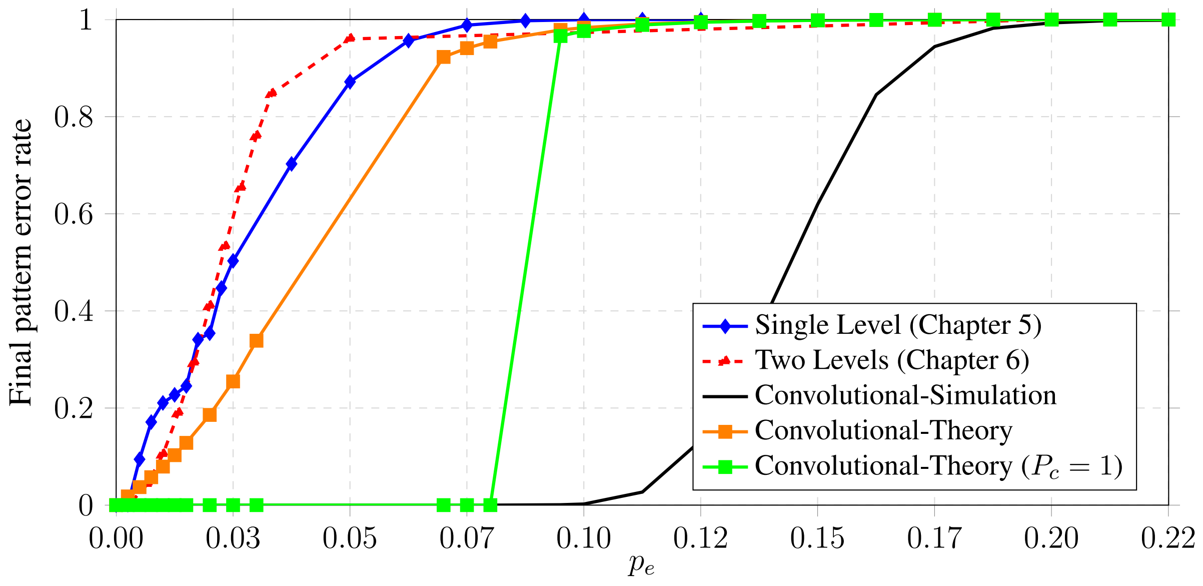


Figure 7.7: Recall error rate and the theoretical bounds for different architectures of network with $n = 400$ pattern neurons and $L = 50$ clusters.

Figure 7.8 shows the final PER for the network with $n = 960$ and $L = 60$ clusters. Comparing the PER with that of a network with $n = 400$ neurons and $L = 50$ clusters, we witness a worse performance. This might seem surprising at first glance since we have increased the network size and the number of clusters. However, the key point in determining the performance of Algorithm 7 is not the number of clusters but the size of the clusters, or the cluster nodes degree distribution, $\rho(x)$. In the former case, each cluster has around 40 pattern neurons while in the latter we have around $n = 80$ neurons in each cluster. If there are too many neurons in one cluster, the chance of getting more than one error in the cluster increases, which means Algorithm 7 would struggle. Thus, increasing the number of clusters will only help when it results in fewer neurons in each cluster.

7.6 Discussion and Final Remarks

In this chapter, we further improved the performance of the recall phase of a neural associative memory that memorize patterns belonging to a subspace. The breakthrough is based on modifying the neural architecture to seek linear constraints among the overlapping sub-patterns of the given patterns. The result is a network that could memorize an exponential

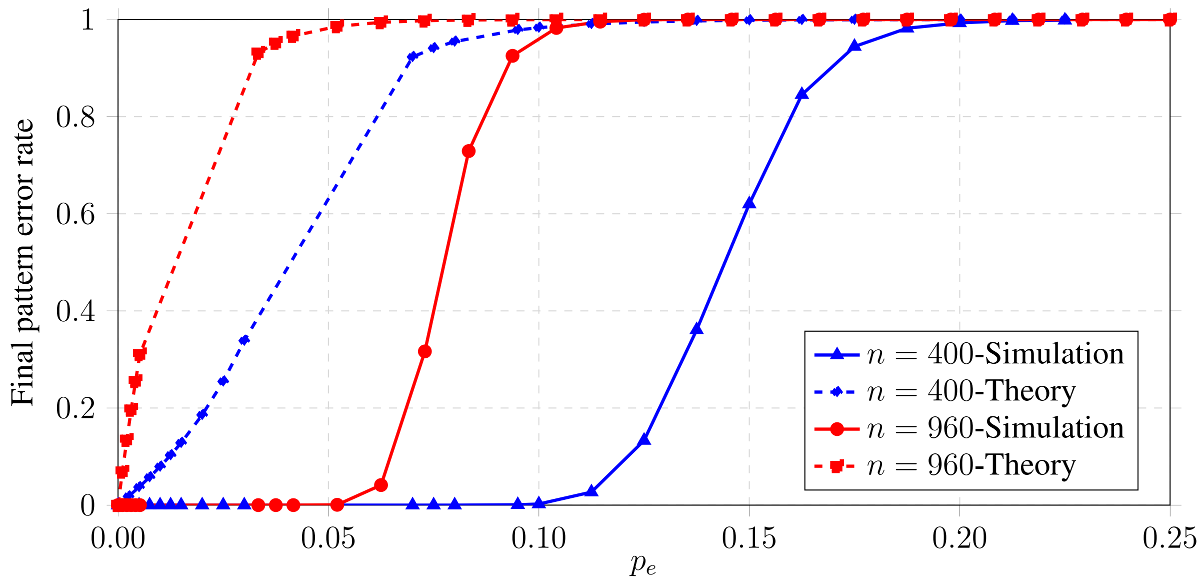


Figure 7.8: Recall error rate and the theoretical bounds for different architectures of network with $n = 800$ pattern neurons and $L = 60$ clusters.

number of patterns while being able to correct a linear number of errors in terms of n, the length of the patterns. Because of the similarities of the new recall algorithm to those used in modern graph based codes, we were able to analyze the average performance of the proposed algorithm using theoretical tools from coding theory.

While the proposed architecture achieves better performance in the recall phase, it is still limited to patterns belonging to a subspace. Extending the model to be able to handle a more general set of patterns is definitely a topic worthy of further investigations. Especially, it would be interesting to evaluate the performance of the proposed method in memorizing natural patterns, e.g., images. In such examples, the patterns in the dataset often do not belong to a subspace. However, in many cases the minor eigenvalues of their correlation matrices is very close to zero, suggesting that the patterns are very close to a low-dimensional embedding. As such, it would be interesting to consider how one could generalize the method proposed in this chapter to address these close-to-subspace cases as well.

Chapter 8

Coupled Neural Associative Memories

In this chapter, we extend the proposed architecture in the previous chapter to achieve even better error correction capabilities. The main idea is to couple several convolutional neural networks to a larger "coupled" architecture. We show that the proposed architecture could still memorize an exponential number of patterns and retrieve the correct pattern in the recall phase if a linear fraction (up to a threshold) of the pattern neurons are corrupted by noise.

Interestingly, the proposed model is very similar to the *spatially coupled codes* in modern coding theory. We will utilize a recently developed method to analyze these spatially coupled codes [61] to theoretically investigate the performance of the proposed model during the recall phase.

It is also interesting to note that the proposed model shares some similarities with the architecture of the visual cortex of macaque brain [62].

8.1 Related Work

As mentioned earlier, the proposed model in this chapter is very similar to spatially-coupled codes on graphs [64]. Specifically, our suggested model is closely related to the spatially-coupled Generalized LDPC code (GLDPC) with Hard Decision Decoding (HDD) proposed in [65]. This similarity helps us borrow analytical tools developed for analyzing such codes [61] and investigate the performance of our proposed neural error correcting algorithm. Similar tools might be helpful in analyzing other neural networks with similar structures as well.

The content of this chapter is joint work with Amin Karbasi and Amin Shokrollahi. It was published in [63].

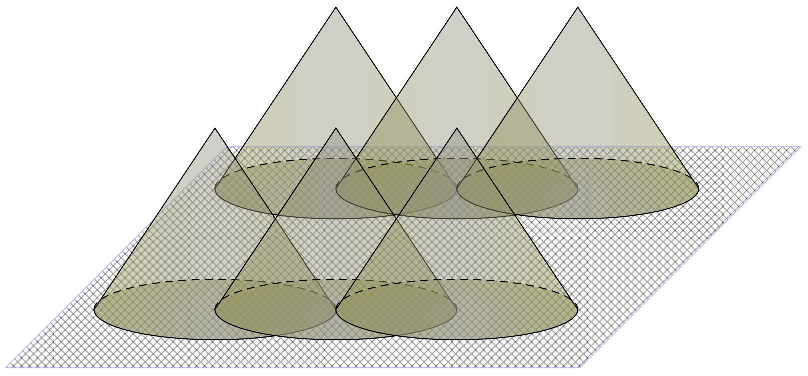


Figure 8.1: Overlapping receptive fields over a 2D pattern. Each "cone" illustrates one cluster and the constraints in that cluster are learned over the domain specified by the cone base.

The proposed approach enjoys the simplicity of message passing operations performed by neurons as compared to the more complex iterative belief propagation decoding procedure of spatially coupled codes [64]. This simplicity may lead to an inferior performance but already allows us to outperform prior error resilient methods suggested for neural associative memories in the literature.

8.2 Problem Formulation and the Model

As before, we will work non-binary neurons in this chapter as well. We are still interested in designing a neural network that is capable of memorizing the patterns in a dataset $\mathcal{X}$. To this end, we break the patterns into smaller pieces/sub-patterns and learn the resulting sub-patterns separately (and in parallel). Furthermore, as our objective is to memorize those patterns that are highly correlated, we only consider a dataset in which the sub-patterns belong to a subspace (or have negligible minor components).

More specifically, and to formalize the problem in a way which is similar to the literature on spatially coupled codes, we divide each pattern into L sub-patterns of the same size and refer to them as *planes*. Within each plane, we further divide the patterns into D *overlapping* clusters, i.e.,, an entry in a pattern can lie in the domain of multiple clusters. We also assume that each element in plane ℓ is connected to at least one cluster in planes $\ell - \Omega, \ldots, \ell + \Omega$ (except at the boundaries). Therefore, each entry in a pattern is connected to $2\Omega + 1$ planes, on average.

An alternative way of understanding the model is to consider 2D datasets, i.e.,, images. In this regard, the overlapping nature of clusters correspond to the overlapping receptive fields over the image (see Figure 8.1), which is similar to the receptive fields in human visual cortex [10]. In addition, rows of the image correspond to planes in our model and clusters are the overlapping "receptive fields" which cover an area composed of neighboring pixels in different rows (planes). The model is shown in Figure 8.2. Our assumptions on strong correlations then translates into assuming strong linear dependencies within each receptive field for all patterns in the dataset.

8.3 Learning Phase

To memorize the patterns, we utilize Algorithm 1 as before, namely we learn a set of vectors that are orthogonal to the sub-patterns in each cluster of each plane. The output of the learning algorithm is an $m_{\ell,d} \times n_{\ell,d}$ matrix $W^{(\ell,d)}$ for cluster d in plane ℓ. The rows of this matrix correspond to the dual vectors and the columns correspond to the corresponding entries in the patterns. Therefore, by letting $\mathbf{x}^{(\ell,d)}$ denote the sub-pattern corresponding to the domain of cluster d of plane ℓ, we have

$$W^{(\ell,d)} \cdot \mathbf{x}^{(\ell,d)} = \mathbf{0}. \tag{8.1}$$

These matrices (i.e.,, $W^{(\ell,d)}$) form the connectivity matrices of our neural graph, in which we can consider each cluster as a bipartite graph composed of *pattern* and *constraint* neurons. The left panel of Figure 8.3 illustrates the model, in which the circles and rectangles correspond to pattern and constraint neurons, respectively. The details of the first plane are magnified in the right panel of Figure 8.3.

Cluster d in plane ℓ contains $m_{\ell,d}$ constraint neurons and is connected to $n_{\ell,d}$ pattern neurons. The constraint neurons do not have any overlaps (i.e., each one belongs only to one cluster) whereas the pattern neurons can have connections to multiple clusters and planes. To ensure good error correction capabilities we aim to keep the neural graph sparse (this model shows significant similarity to some neural architectures in the macaque brain [62]).

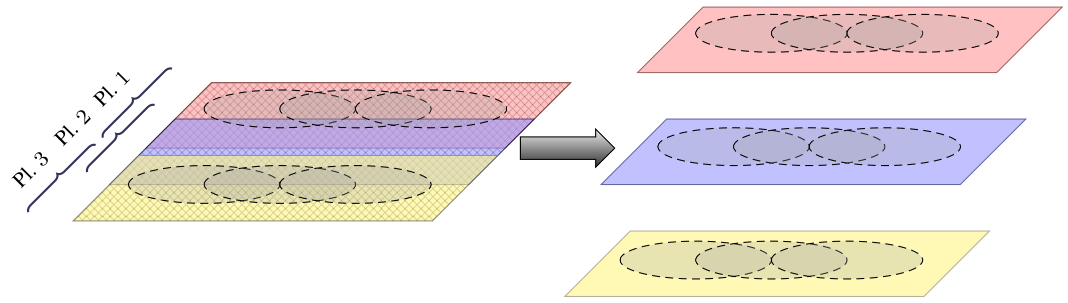


Figure 8.2: Planes and clusters for a 2D pattern, i.e., an image. The domains of the planes are specified in the left figure (i.e., $Pl.\ 1$ indicates the domain of Plane 1).

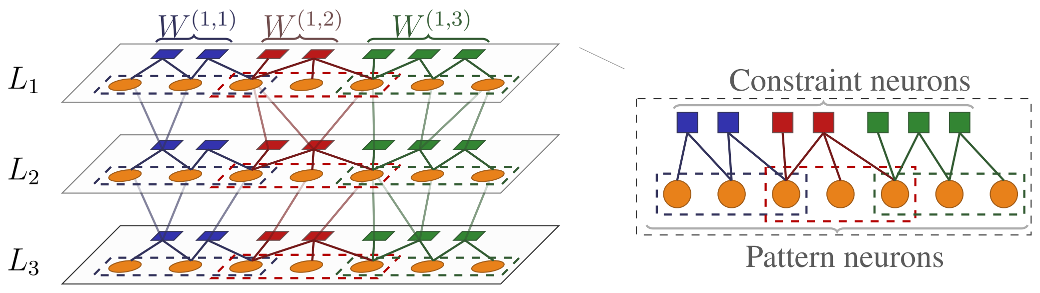


Figure 8.3: A coupled neural associative memory.

We also consider the overall connectivity graph of plane ℓ, denoted by $\widetilde{W}^{(\ell)}$, in which the constraint nodes in each cluster are compressed into one *super constraint node*. Any pattern node that is connected to a given cluster is connected with an (unweighted) edge to the corresponding super constraint node. Figure 8.4 illustrates this graph for plane 1 in Figure 8.3.

8.4 Recall Phase

The recall algorithm in this chapter is very similar to that of the previous chapter. The latter involved a *local* (or intra-cluster) and a *global* (or inter-cluster) algorithm, where the global algorithm repeatedly applies the local algorithm to each cluster in a round robin fashion to ensure good error correction.

Inspired by this boost in the performance, we can stretch the error correction capabilities even further by coupling several neural "planes" with many clusters together, as mentioned earlier. We need to modify the global error correcting algorithm in such a way that it first acts

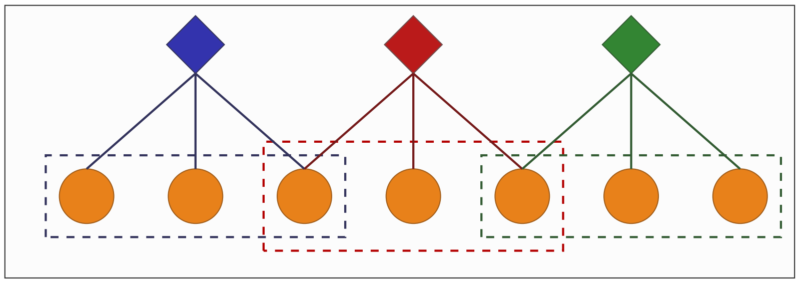


Figure 8.4: A connectivity graph with neural planes and super constraint nodes. It corresponds to plane 1 of Fig. 8.3.

Algorithm 8 Error Correction of the Coupled Network

```
Input: Connectivity matrix ($W^{(\ell,d)}, \forall \ell, \forall d$), iteration $t_{\max}$
Output: Correct memorized pattern $\mathbf{x} = [x_1, x_2, \dots, x_n]$
1: for $t = 1 \to t_{\max}$ do
2:     for $\ell = 1 \to L$ do
3:         for $d = 1 \to D$ do
4:             Apply Algorithm 5 to cluster $d$ of neural plane $\ell$.
5:             Update the value of pattern nodes $\mathbf{x}^{(\ell,d)}$ only if all the constraints in the clustered
               are satisfied.
6:         end for
7:     end for
8: end for
```

upon the clusters of a given plane in each round before moving to the next plane. The whole process is repeated few times until all errors are corrected or a threshold on the number of iterations is reached ($t_{\max}$). Algorithm 8 summarizes the proposed approach.

We consider two variants of the above error correction algorithm. In the first one, called *constrained* coupled neural error correction, we provide the network with some side information during the recall phase. This is equivalent to "freezing" a few of the pattern neurons to known and correct states, similar to spatially-coupled codes [61], [64]. In the case of neural associative memory, the side information can come from the context. For instance, when trying to correct the error in the sentence "The $\underline{c}$at flies", we can use the side information (flying) to guess the correct answer among multiple choices. Without this side information, we cannot tell if the correct answer corresponds to *bat* or *rat*.[1]

In the other variant, called *unconstrained* coupled neural error correction, we perform the error correction without providing any side information. This is similar to many standard recall algorithms in neural networks. In fact, the unconstrained model can be thought of as a very large convolutional network similar to the model proposed in Chapter 7. Thus, the unconstrained model serves as a benchmark to evaluate the performance of the proposed coupled model in this chapter.

Performance Analysis

Let $z^{(\ell)}(t)$ denote the *average* probability of error for pattern nodes across neural plane ℓ in iteration t. Thus, a super constraint node in plane ℓ receives noisy messages from its

[1]The same situation also happens in dealing with erasures, i.e., when trying to fill in the blank in the sentence "The _*at* flies".

neighbors with an average probability $\bar{z}^{(\ell)}$:

$$\bar{z}^{(\ell)} = \frac{1}{2\Omega+1}\sum_{j=-\Omega}^{\Omega} z^{(\ell-j)} \text{ s.t. } z^{(l)} = 0,\ \forall l \notin \{1,\dots,L\}.$$

Our goal is to derive a recursion for $z^{(\ell)}(t+1)$ in terms of $z^{(\ell)}(t)$ and $\bar{z}^{(\ell)}(t)$. To this end, in the graph $\widetilde{W}^{(\ell)}$ let $\lambda_i^{(\ell)}$ and $\rho_j^{(\ell)}$ be the fraction of edges (normalized by the total number of edges in graph $\widetilde{W}^{(\ell)}$) connected to pattern and super constraint nodes with degree i and j, respectively. We define the degree distribution polynomials in plane ℓ from an *edge perspective* as $\lambda^{(\ell)}(x) = \sum_i \lambda_i^{(\ell)} x^{i-1}$ and $\rho^{(\ell)}(x) = \sum_j \rho_i^{(\ell)} x^{j-1}$. Furthermore, let e be the minimum number of errors each cluster can correct with high probability. [2]

Lemma 35. *Let us define $g(z) = 1-\rho(1-z)-\sum_{i=1}^{e-1}\frac{z^i}{i!}\frac{d^i\rho(1-z)}{dz^i}$ and $f(z,p_e) = p_e\lambda(z)$, where e is the number of errors each cluster can correct. Then,*

$$z^{(\ell)}(t+1) = f\left(\frac{1}{2\Omega+1}\sum_{i=-\Omega}^{\Omega} g(\bar{z}^{(\ell-i)}(t)), p_e\right). \tag{8.2}$$

Proof. Without loss of generality, we prove the lemma for the case that each cluster can correct at least two errors with high probability, i.e., $e = 2$. Extending the proof to $e > 2$ will be straightforward.

Let $z^{(\ell)}(t)$ denote the *average* probability of error for pattern nodes across neural plane ℓ and in iteration t. Furthermore, let $\pi^{(\ell)}(t)$ be the *average* probability of a super constraint node in plane ℓ sending a failure message to its neighbors among pattern neurons. We will derive recursive expressions for $z^{(\ell)}(t)$ and $\pi^{(\ell)}(t)$.

A super constraint node in plane ℓ receives noisy messages from its neighbors with an average probability of $\bar{z}^{(\ell)}$, where

$$\bar{z}^{(\ell)} = \frac{1}{2\Omega+1}\sum_{j=-\Omega}^{\Omega} z^{(\ell-j)}$$

with $z^{(i)} = 0$ for $i \leq 0$ and $i > L$.

Let $\pi_i^{(\ell)}$ denote the the probability that a super constraint node with degree i in plane ℓ sends a failure message to one of its neighboring noisy pattern nodes. Then, knowing that each super constraint node (cluster) is capable of correcting at least $e = 2$ errors, $\pi_i^{(\ell)}$ is equal to the probability of receiving two or more noisy messages from *other* pattern neurons,

$$\pi_i^{(\ell)} = 1-\left(1-\bar{z}^{(\ell)}\right)^{i-1} - (i-1)\bar{z}^{(\ell)}\left(1-\bar{z}^{(\ell)}\right)^{i-2}.$$

[2]Simulations in the previous chapter show that clusters can potentially correct $e > 1$ errors with a non-zero probability where e is still a constant, in terms of n, and very small.

Now, letting $\pi^{(\ell)}(t)$ denote the average probability of sending erroneous nodes by super constraint nodes in plane ℓ and in iteration t, we will have

$$\begin{aligned}
\pi^{(\ell)}(t) &= \mathbb{E}\{\pi_i^{(\ell)}\} \\
&= \sum_i \rho_i \pi_i^{(\ell)} \\
&= 1 - \rho(1-\bar{z}^{(\ell)}(t)) - \bar{z}^{(\ell)}(t)\rho'(1-\bar{z}^{(\ell)}(t)),
\end{aligned}$$

where $\rho(z) = \sum_i \rho_i z^{i-1}$ is the super constraint node degree distribution polynomial and $\rho'(z) = d\rho(z)/dz$.

To simplify notations, let us define the function $g(z) = 1 - \rho(1-z) - z\rho'(1-z)$ such that

$$\pi^{(\ell)}(t) = g(\bar{z}^{(\ell)}(t)).$$

Now consider a given pattern neuron with degree j in plane ℓ. Let $z_j^{(\ell)}(t+1)$ denote the probability of sending an erroneous message by this node in iteration $t+1$ over a particular edge to a particular super constraint node. Then, $z_j^{(\ell)}(t+1)$ is equal to the probability of this node being noisy in the first place (p_e) and having all its *other* super constraint nodes sending erroneous messages in iteration t, the probability of which is $\left(\bar{\pi}^{(\ell)}(t)\right)^{j-1}$, where

$$\bar{\pi}^{(\ell)}(t) = \frac{1}{2\Omega+1}\sum_{i=-\Omega}^{\Omega} \pi^{(\ell-i)}(t)$$

is the average probability that a pattern neuron in plane ℓ receives a failure message from its neighboring super constraint neurons.

Now, since $z^{(\ell)}(t+1) = \mathbb{E}\{z_j^{(\ell)}(t+1)\}$, we get

$$\begin{aligned}
z^{(\ell)}(t+1) &= p_e \sum_j \lambda_j \left(\bar{\pi}^{(\ell)}\right)^{j-1} \\
&= p_e \lambda(\bar{\pi}^{(\ell)}) \\
&= p_e \lambda\left(\frac{1}{2\Omega+1}\sum_{i=-\Omega}^{\Omega} g(\bar{z}^{(\ell-i)}(t))\right)
\end{aligned}$$

Again to simplify the notation, let us define the function $f(z, p_e) = p_e\lambda(z)$. This way, we will have the recursion as:

$$z^{(\ell)}(t+1) = f\left(\frac{1}{2\Omega+1}\sum_{i=-\Omega}^{\Omega} g(\bar{z}^{(\ell-i)}(t)), p_e\right).$$

$\square$

The decoding will be successful if $z^{(\ell)}(t+1) < z^{(\ell)}(t),\ \forall \ell$. As a result, we look for the maximum p_e such that

$$f\left(\frac{1}{2\Omega+1}\sum_{i=-\Omega}^{\Omega} g(\bar{z}^{(\ell-i)}(t)), p_e\right) < z^{(\ell)} \text{ for } z^{(\ell)} \in [0, p_e].$$

Let $p_e^{\dagger}$ and p_e^{*} be the maximum p_e's that admit successful decoding for the uncoupled and coupled systems, respectively. To estimate these thresholds, we follow the approach recently proposed in [61] and define a potential function to track the evolution of Equation (8.2). Let $\mathbf{z} = \{z^{(1)}, \dots, z^{(L)}\}$ denote the vector of average probabilities of error for pattern neurons in each plane. Furthermore, let $\mathbf{f}(\mathbf{z}, p_e) : \mathbb{R}^L \to \mathbb{R}^L$ and $\mathbf{g}(\mathbf{z}) : \mathbb{R}^L \to \mathbb{R}^L$ be two component-wise vector functions such that $[\mathbf{f}(\mathbf{z}, p_e)]_i = f(z_i, p_e)$ and $[\mathbf{g}(\mathbf{z})]_i = g(z_i)$, where $f(z_i, p_e)$ and $g(z_i)$ are defined in Lemma 35. Using these definitions, we can rewrite Equation (8.2) in the vector form as [61]:

$$\mathbf{z}(t+1) = A^{\top}\mathbf{f}(A\mathbf{g}(\mathbf{z}(t)), p_e) \tag{8.3}$$

where A is the *coupling matrix* defined as[3]:

$$A = \frac{1}{2\Omega+1}\begin{bmatrix} \overbrace{1 & 1 & \cdots & 1}^{\Omega} & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & \cdots & 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & & & & & & & \\ 0 & 0 & \cdots & 0 & 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

At this point, the potential function of the unconstrained coupled system can be defined as [61]:

$$\begin{aligned} U(\mathbf{z}, p_e) &= \int_C \mathbf{g}'(\mathbf{u})(\mathbf{u} - A^{\top}\mathbf{f}(A\mathbf{g}(\mathbf{u})).d\mathbf{u} \\ &= \mathbf{g}(\mathbf{z})^{\top}\mathbf{z} - G(\mathbf{z}) - F(A\mathbf{g}(\mathbf{z}), p_e) \end{aligned} \tag{8.4}$$

where $\mathbf{g}'(\mathbf{z}) = \text{diag}([g'(z_i)])$, $G(\mathbf{z}) = \int_C \mathbf{g}(\mathbf{u}) \cdot d\mathbf{u}$ and $F(\mathbf{z}) = \int_C \mathbf{f}(\mathbf{u}) \cdot d\mathbf{u}$.

A similar quantity can be defined for the uncoupled (scalar) system as $U_s(z, p_e) = zg(z) - G(z) - F(g(z), p_e)$ [61], where z is the average probability of error in pattern

[3]Matrix A corresponds to the unconstrained system. A similar matrix can be defined for the constrained case.

neurons. The scalar potential function is defined in the way that $U'_s(z, p_e) > 0$ for $p_e \leq p_e^\dagger$. In other words, it ensures that $z(t+1) = f(g(z(t), p_e) < z(t)$ (successful decoding) for $p_e \leq p_e^\dagger$.

Furthermore, let us define $p_e^* = \sup\{p_e | \min(U_s(z, p_e) \geq 0\}$. Thus, in order to find p_e^*, it is sufficient to find the maximum p_e such that $\min\{U_s(z, p_e)\} > 0$ [61]. We will show that the constrained coupled system achieves successful error correction for $p_e < p_e^*$. Intuitively, we expect to have $p_e^\dagger \leq p_e^*$ (side information only helps), and as a result a better error correction performance for the constrained system. Theorem 36 and our experimental result will confirm this intuition later in the chapter.

Let $\Delta E(p_e) = \min_z U_s(z, p_e)$ be the *energy gap* of the uncoupled system for $p_e \in (p_e^\dagger, 1]$ [61]. The next theorem borrows the results of [61] and [64] to show that the constrained coupled system achieves successful error correction for $p_e < p_e^*$.

Theorem 36. *In the constrained system, when $p_e < p_e^*$ the potential function decreases in each iteration. Furthermore, if $\Omega > \frac{\|U''(z,p_e)\|_\infty}{\Delta E(p_e)}$, the only fixed point of Equation (8.3) is 0.*

Proof. The proof of the theorem relies on results from [64] to show that the entries in the vector $\mathbf{z}(t) = [z^{(1)}(t), \ldots, z^{(L)}(t)]$ are non-decreasing, i.e.,,

$$z^{(1)}(t) \leq z^{(2)}(t) \leq \cdots \leq z^{(L)}(t).$$

This can be shown using induction and the fact that the functions $\mathbf{f}(\cdot, p_e)$ and $\mathbf{g}(\cdot)$ are non-decreasing (see the proof of Lemma 22 in [64] for more details).

Then, one can apply the result of Lemma 3 in [61] to show that the potential function of the constrained coupled system decreases in each iteration. Finally, when

$$\Omega > \|U''(\mathbf{z}, p_e)\|_\infty / \Delta E(p_e)$$

one could apply Theorem 1 of [61] to show the convergence of the probability of errors to zero. □

Note that Theorem 36 provides a sufficient condition (on Ω) for the coupled system to ensure it achieves successful error correction for every p_e upto $p_e = p_e^*$. However, the condition provided by Theorem 36 usually requires Ω to be *too large*, i.e., Ω is required to be as large as 1000 to 10000, depending on the degree distributions. Nevertheless, in the next section we show that the analysis is still quite accurate for moderate values of Ω, i.e., $\Omega \simeq 2, 3$, meaning that a system with small coupling parameters could still achieve very good error correction in practice.

8.5 Pattern Retrieval Capacity

The following theorem shows that the number of patterns that can be memorized by the proposed scheme is exponential in n, the pattern size.

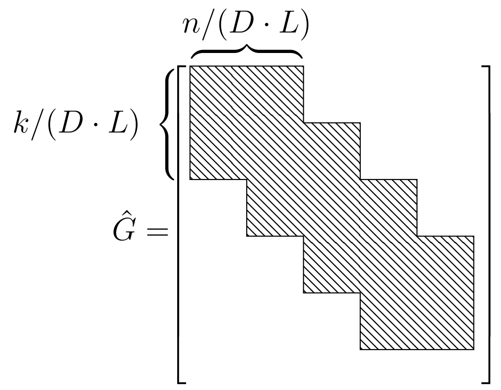
Theorem 37. *Let $a > 1$ be a real number, and $Q > 2$ be an integer. Let n be an integer and $k = \lfloor rn \rfloor$ for some $0 < r < 1$. Then, there exists a set of $C = a^{\lfloor rn \rfloor}$ vectors in $\{0, 1, \ldots, Q-1\}^n$ such that the dimension of the vector space spanned by these vectors is k. This set can be memorized by the proposed algorithm.*

Proof. Since the learning and recall algorithms are not affected by the number of patterns (as long as they come from a subspace), we show that there exist a dataset with an exponential number of patterns that satisfy the requirements of the theorem (i.e. their entries are between 0 and $Q-1$ and the patterns form a subspace). Thus, the proposed algorithm could memorize this dataset and the pattern retrieval capacity could be exponential.

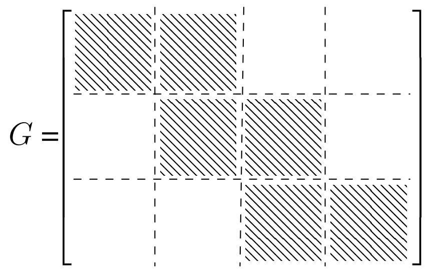
The proof is based on construction: we construct a data set $\mathcal{X}$ with the required properties such that it can be memorized by the proposed neural network. To simplify the notations, we assume all the clusters have the same number of pattern and constraint neurons, denote by $\widetilde{n}_c$ and $\widetilde{m}_c$. In other words, $n_{\ell,d} = \widetilde{n}_c$ and $m_{\ell,d} = \widetilde{m}_c$ for all $\ell = \{1, \ldots, L\}$ and $d = \{1, \ldots, D\}$.

We start by considering a matrix $G \in \mathbb{R}^{k \times n}$, with non-negative integer-valued entries between 0 and $\gamma - 1$ for some $\gamma \geq 2$. We also assume $k = rn$, with $0 < r < 1$.

To construct the dataset, we first divide the columns of G into L sets, each corresponding to the neurons in one plain. Furthermore, in order to ensure that all the sub-patterns within each cluster form a subspace with dimension less than $\widetilde{n}_c$, we propose the following structure for the generator matrix G. This structure ensures that the rank of any sub-matrix of G composed of $\widetilde{n}_c$ columns is less than $\widetilde{n}_c$. In the matrices below, the shaded blocks represent parts of the matrix with *some* non-zero entries. To simplify visualization, let us first define the sub-matrix $\hat{G}$ as the building blocks of G:



Then, G is structured as



where each shaded block represents a random realization of $\hat{G}$.

Now consider a random vector $u \in \mathbb{R}^k$ with integer-valued-entries between 0 and $\upsilon - 1$, where $\upsilon \geq 2$. We construct the dataset by assigning the pattern $\mathbf{x} \in \mathcal{X}$ to be $\mathbf{x} = \mathbf{u} \cdot G$, *if* all the entries of $\mathbf{x}$ are between 0 and $Q - 1$. Obviously, since both $\mathbf{u}$ and G have only non-negative entries, all entries in $\mathbf{x}$ are non-negative. Therefore, we just need to ensure that entries of $\mathbf{x}$ are between 0 and $Q - 1$.

Let ϱ_j denote the j^{th} column of G. Then the j^{th} entry in $\mathbf{x}$ is equal to $x_j = \mathbf{u} \cdot \varrho_j$. Suppose ϱ_j has d_j non-zero elements. Then, we have:

$$x_j = u \cdot \varrho_j \leq d_j(\gamma - 1)(\upsilon - 1)$$

Therefore, letting $d^* = \max_j d_j$, we can choose γ, υ and d^* such that

$$Q - 1 \geq d^*(\gamma - 1)(\upsilon - 1) \tag{8.5}$$

to ensure all entries of x are less than Q.

As a result, since there are υ^k vectors u with integer entries between 0 and $\upsilon - 1$, we will have $\upsilon^k = \upsilon^{rn}$ patterns forming $\mathcal{X}$. Which means $\mathcal{C} = \upsilon^{rn}$, which is exponential in n if $\upsilon \geq 2$. $\square$

8.6 Simulation Results

Since coupling mostly affects the performance of the recall algorithm, in this section we will only investigate the improvement that one can achieve by means of coupling. As such, and for the ease of presentation, we can simply produce these matrices by generating sparse random bipartite graphs and assigning random weights to the connections. Given the weight matrices and the fact that they are orthogonal to the sub-patterns, we can assume w.l.o.g. that in the recall phase we are interested in recalling the all-zero pattern from its noisy version.[4]

We treat the patterns in the database as $2D$ images of size 64×64. More precisely, we have generated a random network with 29 planes and 29 clusters within each plane (i.e.,

[4]The MATLAB code that is used in conducting the simulations mentioned in this chapter is available online at `https://github.com/saloot/NeuralAssociativeMemory`.

$L = D = 29$). Each local cluster is composed of 8×8 neurons and each pattern neuron (pixel) is connected to 2 consecutive planes and 2 clusters within each plane (except at the boundaries). This is achieved by moving the 8×8 rectangular window over the $2D$ pattern horizontally and vertically. The degree distribution of this setting is

$$\lambda = \{0.0011, 0.0032, 0.0043, 0.0722, 0, 0.0054, 0, 0.0841, 0.0032, 0, 0, 0.098, 0, 0, 0, 0.7284\},$$

for degrees 1 to 16, and $\rho_{64} = 1$ and $\rho_j = 0$ for $1 \leq j \leq 63$.

We investigated the performance of the recall phase by randomly generating a $2D$ noise pattern in which each entry is set to ± 1 with probability $p_e/2$ and 0 with probability $1 - p_e$. We then apply Algorithm 8 with $t_{\max} = 10$ to eliminate the noise. Once finished, we declare failure if the output of the algorithm, $\hat{\mathbf{x}}$, is not equal to the pattern $\mathbf{x}$ (assumed to be the all-zero vector).

Figure 8.5 illustrates the final error rate of the proposed algorithm, for the constrained and unconstrained system. For the constrained system, we fixed the state of a patch of neurons of size 3×3 at the four corners of the $2D$ pattern. The results are also compared to the corresponding algorithms in Chapter 5 and 7 (uncoupled systems). In the graph corresponding to the method developed in Chapter 5 (the dashed-dotted curve), there is no clustering while in that of Chapter 7, the network is divided into 50 overlapping clusters all lying on a single plane (the dotted curve). Although clustering improves the performance, it is still inferior to the coupled system with some side information (the solid curve). Note also that even though the same recipe (i.e., Algorithm 5) is used in all approaches, the differences in the architectures has a profound effect on the performance. One also notes the sheer positive effect of network size on the performance (the dotted vs. dashed curves).

Table 8.1 shows the thresholds $p_e^\dagger$ and p_e^* for different values of e. From Figure 8.5 we notice that $p_e^* \simeq 0.39$ and $p_e^\dagger \simeq 0.1$ which is close to the thresholds for $e = 2$ in Table 8.1. This shows that each cluster can most likely correct $e = 2$ errors with hight probability.

Furthermore, note that according to Theorem 36, a sufficient condition for these thresholds to be exact is for Ω to be very large. However, the comparison between Table 8.1 and Figure 8.5 suggests that one can obtain rather exact results even with Ω being rather small.

	$p_e^\dagger$	p_e^*
$e = 1$	0.08	0.18
$e = 2$	0.12	0.36

Table 8.1: Thresholds for the uncoupled ($p_e^\dagger$) and coupled (p_e^*) systems.

Figure 8.6 illustrates how the potential function for uncoupled system (U_s) behaves as a function of z and for various values of p_e. Note that for $p_e \simeq p_e^*$, the minimum value of

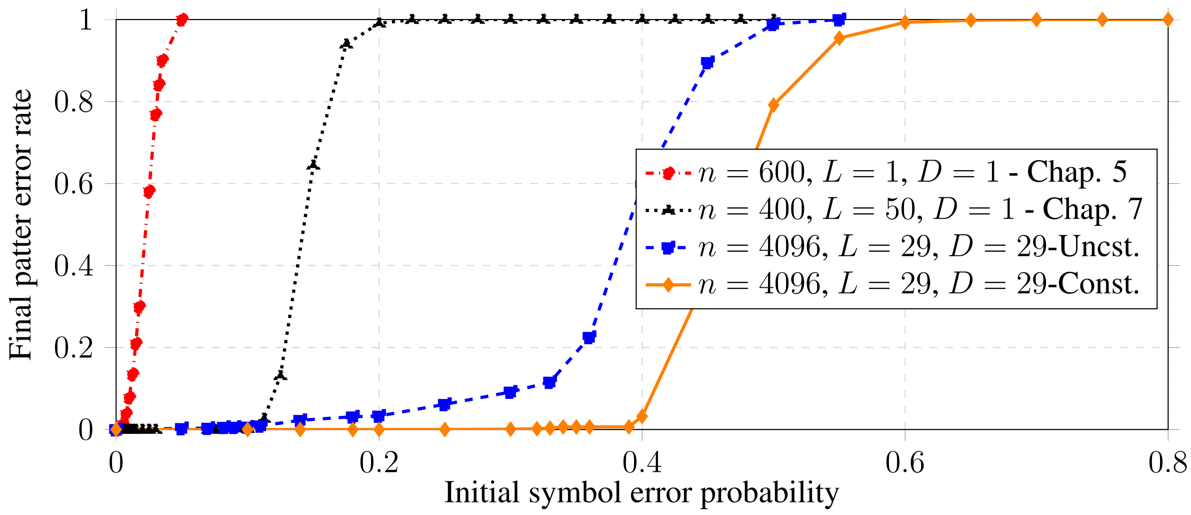


Figure 8.5: The final pattern error probability for the constrained and unconstrained coupled neural systems.

potential reaches zero, i.e. $\Delta_E(p_e^*) = 0$, and for $p_e > p_e^*$ the potential becomes negative for large values of z.

8.7 Discussion and Final Remarks

The architecture proposed in this chapter further improves the performance of the recall phase by expanding the neural graph and exploiting the overlap among neural blocks as well as some side information provided to the network by outside sources. We were able to analyze the average performance of the proposed method using recent developments in the field of spatially coupled codes.

As with the models described in the previous chapters, extending the algorithms to address patterns that are "approximately" in a subspace is definitely among our future research topics.

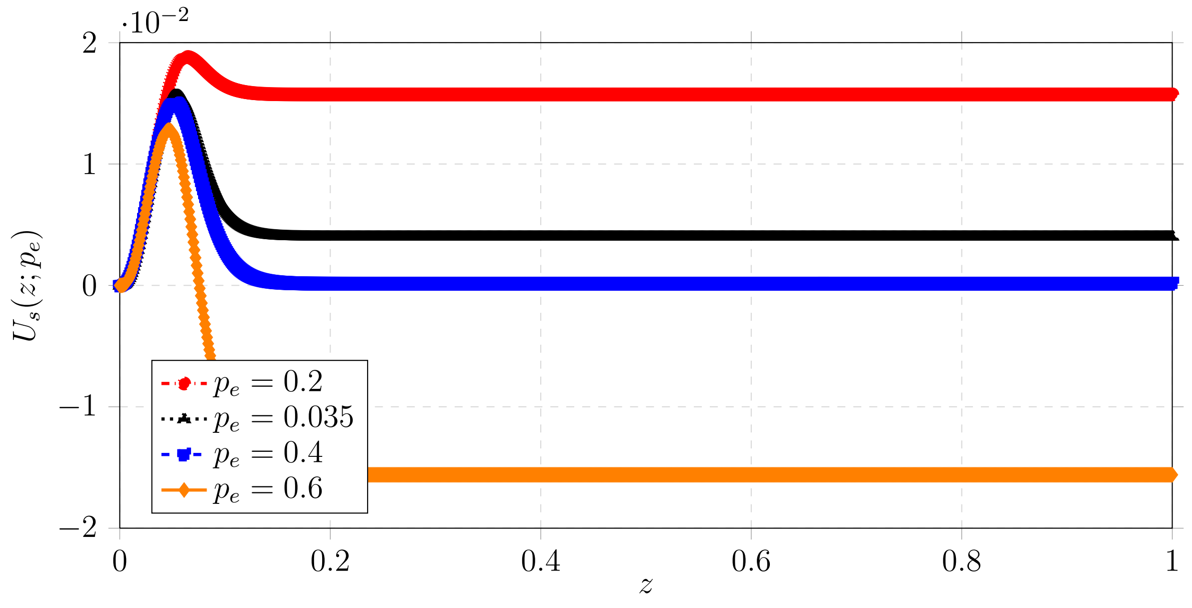


Figure 8.6: The scalar potential function U_s as function of average pattern neurons error probability, z, and different initial symbol error probabilities, p_e.

Part IV

What if the Patterns Do Not Form a Subspace?

Chapter 9

Nonlinear Neural Associative Memories

In the previous chapters, we considered a neural associative model which was based on the idea of learning patterns that form a subspace. In other words, our goal was to look for linear constraints among the patterns. In this chapter we extend this concept to search for nonlinear constraints within the patterns in order to memorize the given data set. More interestingly, we show how a simple trick enables us to re-use the networks that were designed to operate in the linear regime. As a result, we will be able to utilize virtually the same set of learning rules and recall algorithms to implement a nonlinear neural associative memory, which makes the proposed approach more versatile in being able to memorize a broader set of patterns.

9.1 Problem Formulation and the Model

Let $x_1, \dots, x_n$ represent the elements of an n-dimensional input pattern x. Our goal so far has been to find a vector w such that $w^\top \cdot x = w_i x_1 + \cdots + w_n x_n = 0$ (or in general $w^\top \cdot x = c$, for some constant c). As we have seen, there are neural algorithms capable of achieving this goal and we could also make them find a w that is sparse.

Now it is a good point to ask if and how we can find a similar algorithm which identifies non-linear *polynomial* constraints, e.g., $w_1 x_1^2 + w_2 x_1 x_2 + w_3 \sqrt{x_3} = 0$. Interestingly, using a simple trick we can easily transform this problem to the linear case and apply the same learning algorithm we used before to identify non-linear constraints such as the above example.

To start, let us assume that we have a two layer neural network, as shown in Figure 9.1.

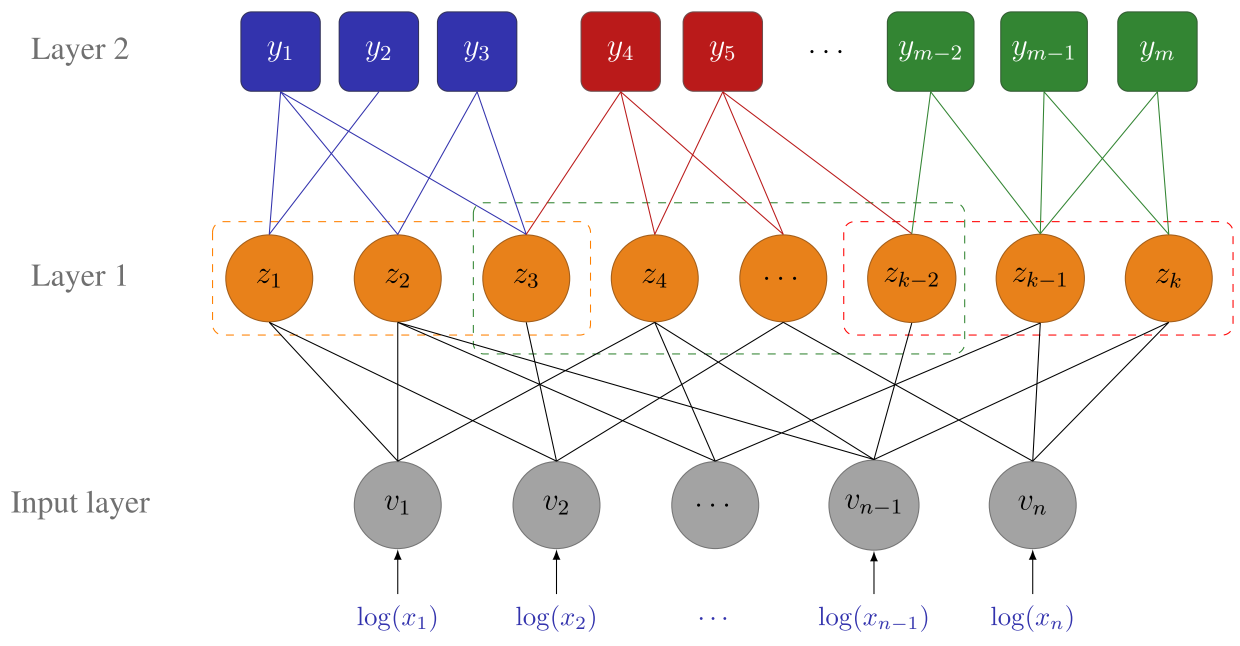


Figure 9.1: The neural network for classifying based on a polynomial kernel.

The connectivity matrices for the first and the second layers are given by W and $\widetilde{W}$, respectively. Now, we give $\ln(x) = [\ln(x_1), \ldots, \ln(x_n)]$ to the input part of the first layer (as opposed to the usual method of feeding the first layer with $x = [x_1, \ldots, x_n]$). Furthermore, let the activation function of the output neurons of the first layer be $f(.) = \exp(.)$.

Therefore, the sate of the output neuron i in the first layer is determined by

$$z_i = f([W \ln(x)]_i) = \exp(\sum_{j=1}^{n} W_{ij} \ln(x_j)) = \prod_{j=1}^{n} x_j^{W_{ij}}$$

Thus, the output of the first layer, which is the input to the second layer, gives us the necessary polynomial terms. Note that constraints are non-linear in the x_j's but they are linear in the z_i's! As a result, we can now apply the standard learning algorithm to identify "linear" constraints in the input of the second layer.

Remark 38. *In order to make the model used in this chapter consistent with the previous chapters, we could consider integer values for x_i's. However, we could slightly generalize our approach by considering real-valued x_i's and then assume that the activation function of the first and second layers map the input to the set $\{0, \ldots, Q-1\}$, as will be discussed shortly. In both cases, we assume that the input layer values given by the vector $(x_1, \ldots, x_n)$*

is mapped to the interval $[0, \infty)$ *so that* $\log(x_i)$ *is well-defined. This mapping is not shown in the model for clarity.*

To simplify further developments, let us fix the following notations:

- The elements of the n-dimensional pattern μ are shown by the vector $x^\mu = [x_1^\mu, \dots, x_n^\mu]$.
- The state of the neurons in the input layer are $v_i = \ln(x_i)$ for $i = 1, \dots, n$.
- The state of the neurons in the first and second layers are denoted by the vectors $z = [z_1, \dots, z_k]$ and $y = [y_1, \dots, y_m]$.
- The activation function of the neurons in the first and second layers are indicated by $f(.)$ and $g(.)$, respectively. Thus, $z_i = f([Wv]_i)$ and $y_j = g([\widetilde{W}z]_j)$.

Note that the messages sent by the neurons in each layer is assumed to be integer-valued as before. As such, the functions $f(.)$ and $g(.)$ are quantized to the range of integers in $\mathcal{Q} = \{0, \dots, Q-1\}$. However, for the sake of brevity we do not show the quantization operation in this section, assuming that the number of quantization levels Q is large enough to be closely approximated by real-valued functions $f(.)$ and $g(.)$.

9.2 Learning Phase

To perform the learning algorithm, we formalize the problem in an optimization framework as before. To this end, we will have

$$\min_{W,\widetilde{W}} E = \sum_{x^\mu \in \mathcal{X}} \|g\left(\widetilde{W}z^\mu\right)\|^2 + \text{sparsity penalty}, \tag{9.1a}$$

subject to

$$\|W_i\|_2 = 1, \ \forall i = 1, \dots, k \ , \tag{9.1b}$$

and

$$\|\widetilde{W}_i\|_2 = 1, \ \forall i = 1, \dots, m \ . \tag{9.1c}$$

In the above equations, W_i and $\widetilde{W}_i$ are the i$^{\text{th}}$ rows of the matrices W and $\widetilde{W}$, respectively, and $z^\mu = f(W\ln(x^\mu))$. The sparsity penalty is a function that approximates $\|.\|_1$ (for W_i and $\widetilde{W}_i$) similar to the one we defined in Chapter 5. Henceforth, we ignore the update term due to the sparsity penalty for brevity and will include it in the final update rule.

As usual, we take the derivative of the objective function with respect to W_{ij} and $\widetilde{W}_{ij}$ to find the update rule for the weights. Furthermore, to deal with the constraints (9.1b) and (9.1c), we can either follow the same approach as Section 5.3 of Chapter 5, namely,

Algorithm 9 Iterative Learning

```
Input: Set of patterns $x^\mu \in \mathcal{X}$ with $\mu = 1, \dots, C$, stopping point $\varepsilon$, Local iteration number,
  $T-1$
Output: $W$ and $\widetilde{W}$
  $t = 1$
  while $\sum_\mu |x^\mu \cdot w(t)|^2 > \varepsilon$ do
    $\hat{W}(1) = \widetilde{W}(t)$
    for $\hat{t} = 1 \to T$ do
      Choose $x(\hat{t})$ at random from patterns in $\mathcal{X}$
      Compute $v(\hat{t}) = \ln(x(\hat{t}))$, $z(\hat{t}) = f(W(t)v(\hat{t}))$ and $y(\hat{t}) = g(\hat{W}(\hat{t})z(\hat{t})$.
      Update $\hat{W}(\hat{t}+1) = \hat{W}(\hat{t}) - \widetilde{\alpha}_t \left(G'(u(\hat{t}))y(\hat{t})z(\hat{t})^\top - \hat{A}(\hat{t})\hat{W}(\hat{t})\right)$
    end for
    $\widetilde{W}(t) = \hat{W}(T)$
    Update $W(t+1) = W(t) - \alpha_t \left(F'(x(t))\widetilde{W}(t)^\top G'(u(t))y(t)x(t)^\top - A(t)W(t)\right)$.
    $t \leftarrow t+1$.
  end while
```

to normalize the rows of the weight matrices in each iteration, or proceed with a simpler approach to include the constraints as a penalty function in the update terms (and equivalently in the original objective functions). We opt for the second approach, which results in the following learning rules:

$$\widetilde{W}(t+1) = \widetilde{W}(t) - \widetilde{\alpha}_t \left(G'(u(t))y(t)z(t)^\top - \widetilde{A}(t)\widetilde{W}(t)\right), \tag{9.2a}$$

$$W(t+1) = W(t) - \alpha_t \left(F'(x(t))\widetilde{W}^\top G'(u(t))y(t)x(t)^\top - A(t)W(t)\right), \tag{9.2b}$$

where $u(t) = \widetilde{W}(t)z(t)$, $G'(.)$ is a diagonal matrix whose entries are the derivative of the function $g(.)$, $F'(.)$ is a diagonal matrix whose entries are the derivative of the function $f(.)$ and $\widetilde{\alpha}_t$ and α_t are small positive learning rates. In addition, the matrices $\widetilde{A}(t)$ and $A(t)$ are two diagonal matrices whose (i,i)-th entries are $1 - \|\widetilde{W}_i\|_2^2$ and $1 - \|W_i\|_2^2$, respectively. Note that the Lagrangian term and the sparsity penalty for ensuring the constraints (9.1b) and (9.1c) are not shown for brevity.

In practice, it is better to update $\widetilde{W}$ several times before adjusting $W(t)$ accordingly. Letting T be the number of local updates for $\widetilde{W}$, the overall procedure is given in Algorithm 9.

9.3 Recall Phase

Given that the nonlinear system is very similar to the linear one, we adopt a similar recall strategy as well. To this end, we will use a convolutional architecture[1] with L clusters, defined by their weight matrices, $W^{(1)}, \dots, W^{(L)}$ and $\widetilde{W}^{(1)}, \dots, \widetilde{W}^{(L)}$. Cluster i has n_i input pattern neurons, k_i neurons in layer 1 and m_i constraint neurons (see Figure 9.1).

The proposed recall algorithm has two phases as before: intra-cluster and inter-cluster. The inter-cluster algorithm is the same as Algorithm 7 of Chapter 7, namely, to sequentially apply the intra-cluster recall algorithm to each cluster and keep the updated neural states only if all constraint neurons in the cluster are satisfied.

The intra-cluster algorithm is a bit different from the previous approaches since we now have two separate layers. As a result, in the recall phase we should send the feedback from constraint neurons in layer 2 to the pattern neurons in the input layer. As before, the constraint neurons send proper feedback when receiving a value that is substantially away from zero. Pattern neurons act upon the received feedback only if the majority of their neighbors tell them to update their state.

However, here the intermediate neurons in layer 1 could play a filtration role. To see how, consider two different scenarios:

1. The intermediate neurons do not filter the messages and send the sign of the feedback they receive as soon as at least one of their neighbors among the constraint neurons signal a constraint violation.

2. The intermediate neurons filter the messages and only alert the pattern neurons if a large portion of their neighbors on the constraints side signal a constraint violation.

We focus on the second approach, as it seems more effective and, additionally, the first one is only a special case of the second method. We first focus on the error correction algorithm in each cluster and then discuss how one could use overlaps among clusters to achieve better error correction performance.

Remark 39. *Note that due to having two layers in the model, a small number of errors in the input layer results in a larger number of errors in the intermediate layer, which makes it difficult for the error correction algorithms to work properly. Nevertheless, if the neural graph is* sparse*, this phenomenon is controlled to some extent, specially since the intermediate layer has more neurons than the the input layer as well. Thus, in a sparse graph a reasonably larger number of errors in the intermediate layer result in a roughly similar* fraction *of errors in the input and the intermediate layers.*

[1]We could use a coupled architecture as well. However, to simplify the notations and the analysis, we stick to the convolutional model.

Error correction within a cluster

Suppose we are interested in correcting errors in a given cluster ℓ with connectivity matrices of $W^{(\ell)}$ in the first layer and $\widetilde{W}^{(\ell)}$ in the second layer. From the learning phase we know that for a memorized pattern x, we have $y^{(\ell)} = g(\widetilde{W}^{(\ell)} z^{(\ell)}) = 0$, where $z^{(\ell)} = f(W^{(\ell)} \ln(x^{(\ell))})$ and $x^{(\ell)}$ is the realization of pattern x onto the cluster ℓ.

Now during the recall phase, we receive a corrupted version of the memorized pattern, i.e., $\ln(x) + e$, where e is the noise vector.[2] We will use the information $y^{(\ell)} = g(\widetilde{W}^{(\ell)} z^{(\ell)}) = 0$ to eliminate the input error e. To start, we will have

$$y(\ell) = g(\widetilde{W}^{(\ell)} z^{(\ell)}) = g(\widetilde{W}^{(\ell)} \tilde{e}^{(\ell)}),$$

where $\tilde{e}^{(\ell)} = W^{(\ell)} e^{(\ell)}$ and $e^{(\ell)}$ is the realization of the input noise e onto the cluster ℓ. Since the patterns in the second layer form a subspace, we can use a similar approach to the one proposed in Algorithm 5 in Chapter 5. To this end, we will use $\text{sign}(y^{(\ell)})$ as the feedback sent by the constraint to the intermediate neurons. Intermediate neurons then decide if they should update their state according to the input feedback. The decision could be based on either the Winner-Take-All (WTA) or the Majority-Voting (MV) approach. Algorithm 10 shows the approach based on the majority voting technique.

Once Algorithm 10 finishes, we move to correcting errors in the input *only if* all the constraints in the second layer are satisfied. Otherwise, we declare a failure. Let $\tilde{z}^{(\ell)}$ denote the state of the intermediate neurons when all the constraints in the second layer of cluster ℓ are satisfied. Now we consider these values as the desired values and compare them with $z^{(\ell)} = f(W^{(\ell)} \ln(\hat{x}^{(\ell)}))$, where $\ln(\hat{x}^{(\ell)}) = \ln(x^{(\ell)}) + e^{(\ell)}$. The difference will provide us with the necessary information in order to eliminate the input error $e^{(\ell)}$. More specifically, since $f(.) = \exp(.)$ in our setting, let $u = \ln(z^{(\ell)}) - \ln(\tilde{z}^{(\ell)}) = W^{(\ell)} \left(\ln(x^{(\ell)}) + e^{(\ell)} - \ln(x^{(\ell)})\right)$.[3] As a result, we face the same situation as that of Chapter 5, where the input patterns formed a subspace. Thus, we can use the same algorithm to eliminate the input error $e^{(\ell)}$. Algorithm 11 summarizes the proposed approach.

In practice, we can perform Algorithms 10 and 11 several times consecutively to achieve a better error correction performance.

[2]Note that we have considered the effective noise to simplify the notations, i.e., noise is added to $\ln(x)$ instead of x. One could also consider a noise model like $\ln(x + e)$, where noise is added to x. Nevertheless, the second model could be rewritten as $\ln(x + e) = \ln(x) + e'$, where e' is the effective noise.

[3]Here we assume that the probability of converging to another pattern which results in the satisfaction of all the constraints is negligible if we start from the vicinity of a memorized pattern. Please refer to the arguments made in the end of Chapter 5 regarding this assumption.

Algorithm 10 Nonlinear Recall Algorithm: Second Layer

```
Input: Connectivity matrices $W^{(\ell)}$, $\widetilde{W}^{(\ell)}$, threshold $\varphi_2$, iteration $t_{\max}$
Output: $z_1^{(\ell)}, z_2^{(\ell)}, \dots, z_{k_\ell}^{(\ell)}$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the weighted input sum $h_i^{(\ell)} = \sum_{j=1}^{k_\ell} \widetilde{W}_{ij}^{(\ell)} z_j^{(\ell)}$, for each
       neuron $y_i^{(\ell)}$ and set:
       $$y_i^{(\ell)} = \begin{cases} 1, & h_i^{(\ell)} < 0 \\ 0, & h_i^{(\ell)} = 0 \\ -1, & \text{otherwise} \end{cases}.$$
3:     Backward iteration: Each neuron $z_j^{(\ell)}$ computes
       $$g_j^{(\ell)} = \frac{\sum_{i=1}^{m_\ell} \widetilde{W}_{ij}^{(\ell)} y_i^{(\ell)}}{\sum_{i=1}^{m_\ell} |\widetilde{W}_{ij}^{(\ell)}|}.$$
4:     Update the state of each intermediate neuron $j$ according to $z_j^{(\ell)} = z_j^{(\ell)} - \text{sign}(g_j^{(\ell)})$
       only if $|g_j^{(\ell)}| > \varphi_2$.
5:     $t \leftarrow t + 1$
6: end for
```

Recall phase: Intra-cluster error correction

Algorithms 10 and 11 correct errors that are within a given cluster and declare failure if they can not eliminate the input noise. However, as we have seen in previous chapters, a cluster on its own does not have much error correction power. This is where the convolutional model comes to the rescue and the overlap among clusters will help improving the performance of the recall phase. To capitalize on the overlapping structure of clusters, we follow a similar approach to that of Chapter 7, namely, execute Algorithms 10 and 11 consecutively and in a round-robin fashion over clusters 1 to L. After each run, we will keep the new state of the pattern neuron in a cluster ℓ if the corresponding constraints are satisfied. Otherwise, the states are rolled back to their original values (i.e., those before performing the latest iteration of the algorithm). The details of the proposed approach are shown in Algorithm 12.

Performance Analysis

To analyze the performance of the proposed algorithm, we follow the same approach as the one in Theorem 33 in Chapter 7. To this end, let $\hat{\lambda}$ and $\hat{\rho}$ be the pattern and constraint de-

Algorithm 11 Nonlinear Recall Algorithm: Second Layer

```
Input: Connectivity matrices $W^{(\ell)}$, threshold $\varphi_1$, iteration $t_{\max}$
Output: $x_1^{(\ell)}, x_2^{(\ell)}, \dots, x_{n_\ell}^{(\ell)}$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the weighted input sum $h_i^{(\ell)} = \sum_{j=1}^{n_\ell} W_{ij}^{(\ell)} \ln(\hat{x}_j^{(\ell)})$, for
       each neuron $z_i^{(\ell)}$ and set:
       $$u_i^{(\ell)} = \begin{cases} 1, & h_i^{(\ell)} < \ln(\tilde{z}_i^{(\ell)}) \\ 0, & h_i^{(\ell)} = \ln(\tilde{z}_i^{(\ell)}) \\ -1, & \text{otherwise} \end{cases}.$$
3:     Backward iteration: Each neuron $x_j^{(\ell)}$ computes
       $$g_j^{(\ell)} = \frac{\sum_{i=1}^{k_\ell} W_{ij}^{(\ell)} u_i^{(\ell)}}{\sum_{i=1}^{m_\ell} |W_{ij}^{(\ell)}|}.$$
4:     Update the state of each pattern neuron $j$ according to $x_j^{(\ell)} = x_j^{(\ell)} - \text{sign}(g_j^{(\ell)})$ only if
       $|g_j^{(\ell)}| > \varphi_1$.
5:     $t \leftarrow t + 1$
6: end for
```

Algorithm 12 Sequential Intra-Cluster Algorithm for Nonlinear Networks

```
Input: $W^{(1)}, \dots, W^{(L)}$ and $\widetilde{W}^{(1)}, \dots, \widetilde{W}^{(L)}$.
Output: $x_1, x_2, \dots, x_n$
1: while There is an unsatisfied $y^{(\ell)}$, for $\ell = 1, \dots, L$ do
2:     for $\ell = 1 \to L$ do
3:         If $y^{(\ell)}$ is unsatisfied, apply Algorithms 10 and 11 consecutively to cluster $\ell$.
4:         If $y^{(\ell)}$ remained unsatisfied, revert the state of pattern neurons connected to cluster
           $\ell$ to their initial state. Otherwise, keep their current states.
5:     end for
6: end while
7: Declare $x_1, x_2, \dots, x_n$ if all $y^{(\ell)}$'s are satisfied. Otherwise, declare failure.
```

gree distributions (from the edge perspective) of the contracted graph $\hat{G}$, where each cluster is represented by a *super constraint node*. The following theorem provides an asymptotic average-case analysis of the proposed approach, which holds if the decision subgraphs for the pattern neurons in graph $\hat{G}$ are tree-like for a depth of τL, where τ is the total number of number of iterations performed by Algorithm 12.

Theorem 40. *Let q be the number of errors in the input layer that each cluster can correct with probability close to 1. Then, under the assumptions that graph $\hat{G}$ grows large and it is chosen randomly with degree distributions given by $\hat{\lambda}$ and $\hat{\rho}$, Algorithm 12 is successful if*

$$p_e\hat{\lambda}\left(1-\sum_{i=1}^{q}\frac{z^{i-1}}{i!}\cdot\frac{d^{i-1}\hat{\rho}(1-z)}{dz^{i-1}}\right) < (1-\epsilon)z,\ for\ z\in[0,p_e], \tag{9.3}$$

where p_e is the input noise probability and $\epsilon > 0$ is very small constant, independent of t.

Proof. The proof is based on the density evolution technique [54]. We prove the theorem without loss of generality for $q = 2$. The extension to $q > 2$ is straightforward. To start, and with some abuse of notation, let $z(t)$ be the average error probability among pattern neurons in iteration t and $\Pi(t)$ be the average probability that a super constraint node sends a failure message, i.e., that it can not correct external errors lying in its domain. Then, the probability that a pattern neuron with degree d_i send an erroneous message to one of its neighbors among the super constraint nodes is equal to the probability that none of its *other* neighboring clusters can correct the error, i.e.,

$$P_i(t) = p_e(\Pi(t))^{d_i-1}.$$

Averaging over d_i we find the average probability of error in iteration t:

$$z(t+1) = p_e\hat{\lambda}(\Pi(t)). \tag{9.4}$$

Now consider a cluster ℓ that contains d_ℓ pattern neurons. This cluster will *not* send a failure message to a noisy neighboring pattern neuron with probability

1. 1, if it is not connected to more than $q-1=1$ *other* noisy pattern neurons (recall that here $q=2$).
2. 0, otherwise.

Thus, we obtain

$$\Pi^{(\ell)}(t) \quad = \quad 1-(1-z(t))^{d_\ell-1}-\binom{d_\ell-1}{1}z(t)(1-z(t))^{d_\ell-2}. \tag{9.5}$$

Averaging over d_ℓ we obtain

$$\Pi(t) = \mathbb{E}_{d_\ell}\left(\Pi^{(\ell)}(t)\right) = 1 - \hat{\rho}(1-z(t)) - z(t)\hat{\rho}'(1-z(t)), \tag{9.6}$$

where $\hat{\rho}'(z)$ is the derivative of the function $\hat{\rho}(z)$ with respect to z.

Equations (9.4) and (9.6) will give us the value of $z(t+1)$ as a function of $z(t)$. We can then calculate the final symbol error rate as $z(\infty)$. For $z(\infty) \to 0$, it is sufficient to have $z(t+1) < (1-\epsilon)z(t)$, which proves the theorem. □

Note that as in Chapter 7, when graph $\hat{G}$ is constructed randomly according to the given degree distribution, then as the graph size grows, the decision subgraph will become a tree with probability close to 1. Furthermore, and as mentioned in previous chapters, it must be pointed out that the theoretical results derived in this chapter show that *if* the neural connectivity matrix has certain properties then the theoretical estimates on the recall performance will be tighter. In general, the closer the neural graph is to an *expander* (see Appendix 5.B in Chapter 5), the better the recall performance will be.

Remark 41. *In Chapter 7, we computed a theoretical estimate on q, the number of errors that each cluster can correct with probability close to 1 in a convolutional neural architecture. We showed that with high probability, $q \geq 1$.*

Deriving such a theoretical estimate is more difficult for the nonlinear architecture due to having the intermediate layer and the quantization operation. In such a circumstance, we could use numerical approaches to find a tight estimate of q for the nonlinear neural associative memories.

9.4 Final Remarks

The proposed nonlinear architecture in this chapter is virtually the same as the convolutional model we introduced in Chapter 7 to learn linear constraints within the patterns. Nevertheless, a simple trick enabled us to design a network that looks for nonlinear constraints within the given patterns. We also introduced simple learning and recall algorithms to perform the task of neural association. The performance of the proposed recall algorithm can be assessed using similar techniques as those introduced in the previous chapter.

Nevertheless, in practical situation one must be aware of the role of the number of quantization levels, Q, as it might affect the way we calculate the overall recall error rate. In the nonlinear setting where we have two layers, some of the information about the input noise might be lost in the intermediate layer due to the quantization process during the recall process, especially if Q is rather low.

More specifically, at the intermediate layer, we have $z = f(W\ln(x))$, where f is a nonlinear function that approximates $\exp(.)$. At this point, the output is again quantized to the range $\mathcal{Q}$ before being passed to the final layer. The quantization might create an issue for the recall phase though, since the noise in the input layer could be resolved during the quantization in the intermediate layer. For instance, if we have a noise vector $e = [1, 0, \dots, 0]$, then if the first column of W, W_1, has relatively small values, then the quantized version of $z = f(W(v + e))$ would be the same as the quantized version of $f(Wv)$.

As a result, some of the noise is eliminated during the quantization process, without the help of the proposed recall algorithm. This phenomenon is good news for the constraint neurons in the second layer. However, it might make the algorithm unable to obtain the correct input pattern in the *input layer* since some of the information about the noise in the input layer is lost in the quantization process. In such a circumstance, we either have to increase the number of quantization levels or we could calculate the recall error rate from the intermediate layer, i.e., how many errors in the intermediate layer can be corrected. In that case, Theorem 33 in Chapter 7 provides a theoretical estimate on the number of errors that the second layer could correct.

The latter scenario makes sense in certain cases, since, in effect, the state of the intermediate neurons are the patterns that are eventually *memorized* by the network. In this case, one can think of the first layer as a feature extraction layer and the second layer as the effective neural associative memory which is memorizing *features* regarding given patterns, rather than the original patterns themselves.

Part V

Other Extensions and Applications

Chapter 10

Noise-Enhanced Associative Memories

The model that was developed in the previous chapters allowed us to reliably memorize an exponential number of structured patterns and being able to tolerate a linear fraction of errors during the recall phase. Although these designs correct external errors in recall, they assume neurons that compute noiselessly, in contrast to the highly variable nature of neurons in a real neuronal network.

In this chapter, we consider neural associative memories with noisy internal computations and analytically characterize the performance of the recall phase in such conditions. We will show that in our model, as long as the internal noise level is below a specified threshold, the error probability in the recall phase can be made exceedingly small. More surprisingly, we show that internal noise actually improves the performance of the recall phase. Computational experiments lend additional support to our theoretical analysis. This work suggests a functional benefit to noisy neurons in biological neuronal networks, as have been also noted previously in other models as well [6].

The content of this chapter is joint work with Amin Karbasi, Amin Shokrollahi and Lav R. Varshney. It was published in [66].

10.1 Problem Formulation and the Model

Notation and definitions

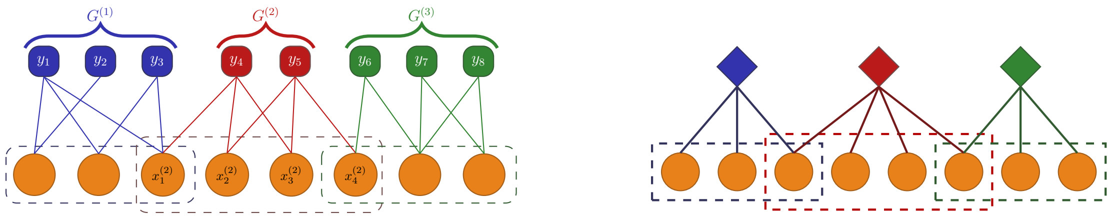
The model considered in this chapter is the same as before, namely, neurons assume integer-valued states from the set $\mathcal{Q} = \{0, \dots, Q-1\}$. However, here the neurons are noisy. As such, a neuron updates its state based on those of its neighbors $\{s_i\}_{i=1}^n$ as follows. It first computes a weighted sum $h = \sum_{i=1}^n w_i s_i + \zeta$, where w_i denotes the weight of the input link from s_i and ζ is the *internal noise*. Each neuron updates its state by applying a nonlinear function $f : \mathbb{R} \to \mathcal{Q}$ to h.

The neural graph and the associative memory model remains the same as before. Since the focus in this chapter is on recalling patterns with strong *local correlation*, we consider the convolutional model we introduced in Chapter 7. To recap, we divide the entries of each pattern x into L *overlapping* sub-patterns of lengths $n_1, \dots, n_L$, as shown in Figure 10.1a. We denote the i^{th} sub-pattern by $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_{n_i}^{(i)})$. The local correlations are assumed to be in the form of subspaces, i.e., the sub-patterns $x^{(i)}$ form a subspace of dimension $k_i < n_i$. We capture the local correlations by learning the matrix orthogonal to subspace formed by sub-patterns. Let $W^{(i)}$ be the matrix that is orthogonal to all the sub-patterns that lie within the domain of cluster i, i.e., $W^{(i)} \cdot x^{(i)} = 0$ for all patterns x in the dataset $\mathcal{X}$. For the forthcoming asymptotic analysis, we also need the *contracted graph* $\widetilde{G}$ the connectivity matrix of which is denoted $\widetilde{W}$ and has size $L \times n$. This is a bipartite graph in which the constraints in each cluster are represented by a single neuron, called super constraint node. Thus, if pattern neuron x_j is connected to cluster i, we set $\widetilde{W}_{ij} = 1$. Otherwise, we have $\widetilde{W}_{ij} = 0$. We also define the degree distribution from an *edge perspective* over $\widetilde{G}$. To this end, we define $\widetilde{\lambda}(z) = \sum_j \widetilde{\lambda}_j z^{j-1}$ and $\widetilde{\rho}(z) = \sum_j \widetilde{\rho}_j z^{j-1}$ where $\widetilde{\lambda}_j$ (resp., $\widetilde{\rho}_j$) equals the fraction of edges that connect to pattern (resp., cluster) nodes of degree j.

Noise model

There are two types of noise in our model: external and internal. To make a distinction, we refer to the former type as *external errors* and to the latter type as *internal noise*. As mentioned earlier, a neural network should be able to retrieve a memorized pattern x from its corrupted version y due to external errors. We assume that the external error is an additive vector of size n, denoted by z, satisfying $y = x + z$, whose entries assume values independently from $\{-1, 0, +1\}$ with corresponding probabilities $p_{-1} = p_{+1} = p_e/2$ and $p_0 = 1 - p_e$.[1] We denote by $z^{(i)}$, the realization of the external error on the sub-pattern $x^{(i)}$.

[1]As noted in the previous chpaters, the proposed model can also deal with higher noise values. The ± 1 noise model has been adopted for its simplicity and the error correction speed.


$G^{(1)}$
$G^{(2)}$
$G^{(3)}$
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8
$x_1^{(2)}$ $x_2^{(2)}$ $x_3^{(2)}$ $x_4^{(2)}$

(a) Bipartite graph G. (b) Contraction graph $\widetilde{G}$.

Figure 10.1: The proposed neural associative memory with overlapping clusters.

Note that due to the subspace assumption, $W \cdot y = W \cdot z$ and $W^{(i)} \cdot y^{(i)} = W^{(i)} \cdot z^{(i)}$ for all i.

Neurons also suffer from internal noise. We consider a bounded noise model, i.e., a random number uniformly distributed in the intervals $[-\upsilon, \upsilon]$ and $[-\nu, \nu]$ for the pattern and constraint neurons, respectively ($\upsilon, \nu < 1$).

The main goal of recall is to remove the external error z and obtain the desired pattern x as the true states of the pattern neurons. When the computation of neurons is noiseless, this task can be achieved by exploiting the fact we have chosen the set of patterns $x \in \mathcal{X}$ to satisfy the set of constraints $W^{(i)} \cdot x^{(i)} = 0$. However, it is not clear how to accomplish this objective when the neural computations are noisy. Rather surprisingly, we show that eliminating external errors is not only possible in the presence of internal noise, but that neural networks with moderate internal noise demonstrate better resilience against external noise.

Related Work

Reliably storing information in memory systems constructed completely from unreliable components is a classical problem in fault-tolerant computing [67–69]. Although direct comparison is difficult since notions of circuit complexity are slightly different, our work also demonstrates that associative memory architectures can store information reliably despite being constructed from unreliable components.

The positive effect of internal noise has been witnessed previously in associative memory models with stochastic update rules (see for instance [6] and [13]). However, the proposed framework in this thesis differs from previous approaches in three key aspects:

1. First and foremost, the model that we use is different from previous methods (namely, non-binary neurons and correlated patterns). This makes the extension of previous analysis nontrivial to our model since they rely on the randomness of the patterns.

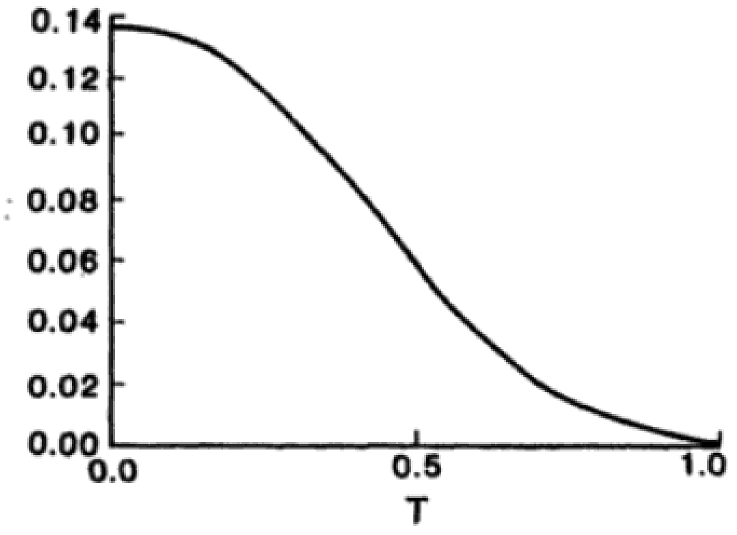


Figure 10.2: The normalized number of memorized patterns as a function of the normalized noise parameter (temperature), T [6]. The normalization is done with respect to the number of neurons in the graph, n.

2. Secondly, and maybe most importantly, pattern retrieval capacity of the network in previous approaches is a *decreasing* function of the internal noise (see Figure 6.1 in [6], which is replicated in Figure 10.2 for convenience). More specifically, although increasing the internal noise helps us correct more external errors, it also reduces the number of patterns that can be memorized. However, in the proposed framework here, the amount of internal noise does not have any effect on the pattern retrieval capacity (up to a threshold of course). Furthermore, a fair amount of internal noise improves the recall performance, as noticed in previous work as well.

3. Finally, the noise model that we use in this thesis is different from the mainstream work in that we use a bounded noise model, rather than the Gaussian model. More specifically, we consider a particular type of noise that is uniformly distributed in a bounded interval.[2] As a result we can show that when the noise is fairly limited, a network with properly chosen update thresholds can achieve *perfect* recall, despite the presence of internal noise.

[2]We believe that the analysis in this thesis can be extended to any noise model with bounded values.

10.2 Recall Phase

To efficiently deal with the external errors in the associative memory, we use a combination of Algorithms 13 and 14, which are the stochastic (noisy) versions of Algorithms 5 and 7. The role of Algorithm 13 is to correct at least one single external error in each cluster. Without overlaps between clusters, the error resilience of the network is limited. Algorithm 14 exploits overlaps: it helps clusters with external errors recover their correct states by using the reliable information from clusters that do not have external errors. The error resilience of the resulting combination thereby drastically improves.

Algorithm 13 performs a series of forward and backward iterations in each cluster $G^{(l)}$ to remove (at least) one external error from its input domain. At each iteration, the pattern neurons locally decide whether to update their current state: if the amount of feedback received by a pattern neuron exceeds a threshold, the neuron updates its state, and remains as is, otherwise. By abuse of notation, let us denote messages transmitted by pattern node i and constraint node j at round t by $x_i(t)$ and $y_j(t)$, respectively. In round 0, the pattern nodes are initialized by a pattern $\hat{x}$, sampled from the dataset $\mathcal{X}$, perturbed by external errors z, i.e.,, $x(0) = \hat{x} + z$. Thus, for cluster ℓ we have $x^{(\ell)}(0) = \hat{x}^{(\ell)} + z^{(\ell)}$, where $z^{(\ell)}$ is the realization of noise on sub-pattern $x^{(\ell)}$.

In round t, the pattern and constraint neurons update their states based on feedback from neighbors. However, neural computations are faulty and, therefore, the decisions of the neurons are not always reliable. To minimize the effect of the internal noise, we opt for the following update rule for pattern node i in cluster ℓ:

$$x_i^{(\ell)}(t+1) = \begin{cases} x_i^{(\ell)}(t) - \text{sign}(g_i^{(\ell)}(t)), & \text{if } |g_i^{(\ell)}(t)| \geq \varphi \\ x_i^{(\ell)}(t), & \text{otherwise,} \end{cases} \tag{10.1}$$

where φ is the update threshold and $g_i^{(\ell)}(t) = \frac{\left((\text{sign}(W^{(\ell)})^\top \cdot y^{(\ell)}(t)\right)_i}{d_i} + u_i$. Here, d_i is the degree of pattern node i, $y^{(\ell)}(t) = [y_1^{(\ell)}(t), \dots, y_{m_\ell}^{(\ell)}(t)]^\top$ is the vector of messages transmitted by the constraint neurons in cluster ℓ, and u_i is the random noise affecting pattern node i. Basically, the term $g_i^{(\ell)}(t)$ reflects the (average) belief of constraint nodes connected to pattern neuron i about its correct value. If $g_i^{(\ell)}(t)$ is larger than a specified threshold φ it means that most of the connected constraints suggest that the current state $x_i^{(\ell)}(t)$ is not correct, hence, a change should be made. Of course, this average belief is diluted by the internal noise of neuron i. As mentioned earlier, u_i is uniformly distributed in the interval $[-\upsilon, \upsilon]$, for some $\upsilon < 1$. On the constraint side, the update rule we choose is

$$y_i^{(\ell)}(t) = f(h_i^{(\ell)}(t), \psi) = \begin{cases} +1, & \text{if } h_i^{(\ell)}(t) \geq \psi \\ 0, & \text{if } -\psi \leq h_i^{(\ell)}(t) \leq \psi \\ -1, & \text{otherwise,} \end{cases} \tag{10.2}$$

where ψ is the update threshold and $h_i^{(\ell)}(t) = \left(W^{(\ell)} \cdot x^{(\ell)}(t)\right)_i + v_i$. Here, $x^{(\ell)}(t) = [x_1^{(\ell)}(t), \dots, x_{n_\ell}^{(\ell)}(t)]^\top$ is the vector of messages transmitted by the pattern neurons and v_i is the random noise affecting node i. As before, we consider a bounded noise model for v_i, i.e.,, it is uniformly distributed in the interval $[-\nu, \nu]$ for some $\nu < 1$.

Algorithm 13 Intra-Module Error Correction

```
Input: Training set $\mathcal{X}$, thresholds $\varphi, \psi$, iteration $t_{\max}$
Output: $x_1^{(\ell)}, x_2^{(\ell)}, \dots, x_{n_\ell}^{(\ell)}$
1: for $t = 1 \to t_{\max}$ do
2:     Forward iteration: Calculate the input $h_i^{(\ell)} = \sum_{j=1}^{n_\ell} W_{ij}^{(\ell)} x_j^{(\ell)} + v_i$, for each neuron $y_i^{(\ell)}$
       and set $y_i^{(\ell)} = f(h_i^{(\ell)}, \psi)$.
3:     Backward iteration: Each neuron $x_j^{(\ell)}$ computes
           $g_j^{(\ell)} = \frac{\sum_{i=1}^{m_\ell} \text{sign}(W_{ij}^{(\ell)}) y_i^{(\ell)}}{\sum_{i=1}^{m_\ell} \text{sign}(|W_{ij}^{(\ell)}|)} + u_i.$
4:     Update the state of each pattern neuron $j$ according to $x_j^{(\ell)} = x_j^{(\ell)} - \text{sign}(g_j^{(\ell)})$ only if
       $|g_j^{(\ell)}| > \varphi$.
5: end for
```

Algorithm 14 Sequential Inter-Module Error Correction Algorithm

```
Input: $\widetilde{G}, G^{(1)}, G^{(2)}, \dots, G^{(L)}$.
Output: $x_1, x_2, \dots, x_n$
1: while there is an unsatisfied $v^{(\ell)}$ do
2:     for $\ell = 1 \to L$ do
3:         If $v^{(\ell)}$ is unsatisfied, apply Algorithm 13 to cluster $G^{(l)}$.
4:         If $v^{(\ell)}$ remained unsatisfied, revert the state of pattern neurons connected to $v^{(\ell)}$ to
           their initial state. Otherwise, keep their current states.
5:     end for
6: end while
7: Declare $x_1, x_2, \dots, x_n$ if all $v^{(\ell)}$'s are satisfied. Otherwise, declare failure.
```

Recall Performance Analysis

Now let us analyze the recall error performance. The following lemma shows that if φ and ψ are chosen properly, then in the absence of external errors the constraints remain satisfied and internal noise cannot result in violation of the constraints. This is a crucial property for

Algorithm 14 to work as it makes it possible to tell if a cluster has successfully eliminated external errors (Step 4 of algorithm) by merely checking the satisfaction of all constraint nodes.

Lemma 42. *In the absence of external errors, the probability that a constraint neuron (resp. pattern neuron) in cluster ℓ makes a wrong decision due to its internal noise is given by $\pi_0^{(\ell)} = \max\left(0, \frac{\nu-\psi}{\nu}\right)$ (resp. $P_0^{(\ell)} = \max\left(0, \frac{\upsilon-\varphi}{\upsilon}\right)$).*

Proof. To calculate the probability that a constraint node makes a mistake when there is no external noise, consider constraint node i whose decision parameter will be

$$h_i^{(\ell)} = \left(W^{(\ell)} \cdot x^{(\ell)}\right)_i + v_i = v_i.$$

Therefore, the probability of making a mistake will be

$$\begin{aligned} \pi_0^{(\ell)} &= \Pr\{|v_i| > \psi\} \\ &= \max\left(0, \frac{\nu - \psi}{\nu}\right). \end{aligned} \tag{10.3}$$

Thus, to make $\pi_0^{(\ell)}$ equal to 0 we will select $\psi > \nu$.[3] So from now on, we assume

$$\pi_0^{(\ell)} = 0. \tag{10.4}$$

Now knowing that the constraint will not send any non-zero messages in absence of external noise, we focus on the pattern neurons in the same circumstance. A given pattern node $x_j^{(\ell)}$ will receive a zero from all its neighbors among the constraint nodes. Therefore, its decision parameter will be $g_j^{(\ell)} = u_j$. As a result, a mistake could happen if $|u_j| \geq \varphi$. The probability of this event is given by

$$\begin{aligned} P_0^{(\ell)} &= \Pr\{|u_i| > \varphi\} \\ &= \max\left(0, \frac{\upsilon - \varphi}{\varphi}\right). \end{aligned} \tag{10.5}$$

Therefore, to make $P_0^{(\ell)}$ go to zero, we must select $\varphi \geq \upsilon$. □

[3]Note that this might not be possible in all cases since, as we will see later, the minimum absolute value of network weights should be at least ψ. Therefore, if ψ is too large we might not be able to find a proper set of weights.

In the remainder of the chapter, we assume $\varphi > \upsilon$ and $\psi > \nu$ so that $\pi_0^{(\ell)} = 0$ and $P_0^{(\ell)} = 0$. However, it is still possible that an external error combined with the internal noise makes the neurons go to a wrong state.

In light of the above lemma and based on our neural architecture, we can prove the following surprising result. We show that in the asymptotic regime (as the number of iterations of Algorithm 14 goes to infinity), a neural network with internal noise outperforms the one without. Let us define the fraction of errors corrected by the noiseless and noisy neural network (parametrized by υ and ν) after T iterations of Algorithm 14 by $\Lambda(T)$ and $\Lambda_{\upsilon,\nu}(T)$, respectively. Note that both $\Lambda(T) \leq 1$ and $\Lambda_{\upsilon,\nu}(T) \leq 1$ are non-decreasing sequences of T. Hence, their limits as T goes to infinity are well defined: $\lim_{T\to\infty} \Lambda(T) = \Lambda^*$ and $\lim_{T\to\infty} \Lambda_{\upsilon,\nu}(T) = \Lambda^*_{\upsilon,\nu}$.

Theorem 43. *Let us choose φ and ψ so that $\pi_0^{(\ell)} = 0$ and $P_0^{(\ell)} = 0$ for all $\ell \in \{1, \ldots, L\}$. For the same realization of external errors, we have $\Lambda^*_{\upsilon,\nu} \geq \Lambda^*$.*

Proof. We first show that the noisy neural network can correct any external error pattern that the noiseless counterpart can correct in the limit of $T \to \infty$. The idea is that if the noiseless decoder succeeds, then there is a non-zero probability P that the noisy decoder succeeds in a given round as well (corresponding to the case that noise values are rather small). Since we do not introduce new errors during the application of Algorithm 14, the number of errors in the new rounds are smaller than or equal to the previous round, hence, the probability of success is lower bounded by P. If we apply Algorithm 14 T times, then the probability of correcting the external errors at the end of round T is $P + P(1-P) + \cdots + P(1-P)^{T-1} = 1-(1-P)^T$.[4] Since $P > 0$, for $T \to \infty$ this probability tends to 1.

Now, we turn our attention to the cases where the noiseless network fails in eliminating external errors and show that there exist external error patterns, called *stopping sets*, for which the noisy decoder is capable of eliminating them while the noiseless network has failed. Assuming that each cluster can eliminate i external errors in their domain and in absence of internal noise[5], stopping sets correspond to external noise patterns for which each cluster has more than i errors. Then Algorithm 14 can not proceed any further. However, in the noisy network, there is a chance that in one of the rounds, the noise acts in our favor and the cluster could correct more than i errors. This is reflected in Figure 10.4 as well, where the value of P_{c_i}, the probability that each cluster can correct i external errors, is larger when the network is noisy. In this case, if the probability of getting out of the stopping set in the noisy network is P in each round, for some $P > 0$, then a similar argument to the previous

[4]Note that since the noiseless decoder is assumed to succeed, P depends on the noise values. Since we have assumed an $i.i.d.$ noise model, the probability of success in trials are independent from each other.

[5]From Figure 10.4, $i = 2$ for our case.

case shows that the overall probability of success tends to 1 when $T \to \infty$ (note that $P = 0$ for the noiseless network per definition of the stopping sets). This concludes the proof. $\square$

The key idea behind Theorem 43 is that one can try several times to eliminate external errors ($T \to \infty$) but must be careful not to introduce any new errors during the correction process (as made sure by Step 4 of Algorithm 14).

Figure 10.3a illustrates an example of a stopping set over the graph $\widetilde{G}$ in our empirical studies. In the figure, only the nodes corrupted with external noise are shown for clarity. Pattern neurons that are connected to at least one cluster with a single error are colored blue and other pattern neurons are colored red. Figure 10.3b illustrates the same network but after a sufficient number of decoding iterations which results in the algorithm to get stuck. Obviously, we have a stopping set in which no cluster has a single error and the algorithm can not proceed further since $P_{c_i} \simeq 0$ for $i > 1$ in a noiseless architecture. Thus, the external error can not get corrected.

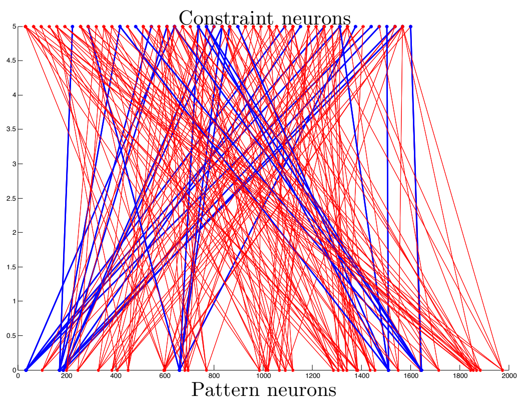
Consequently, the above theorem shows that the supposedly harmful internal noise will help Algorithm 14 to avoid some of such stopping sets. Interestingly, an "unreliable" neural circuit in which $\upsilon = 0.6$ could easily get out of the stopping set shown in Figure 10.3b and correct all of the external errors.

In addition, and rather interestingly, our empirical experiments show that in certain scenarios, even the running time improves when using a noisy network as the number of iterations of Algorithm 14 required to eliminate the external errors is reduced when the network has some internal noise.

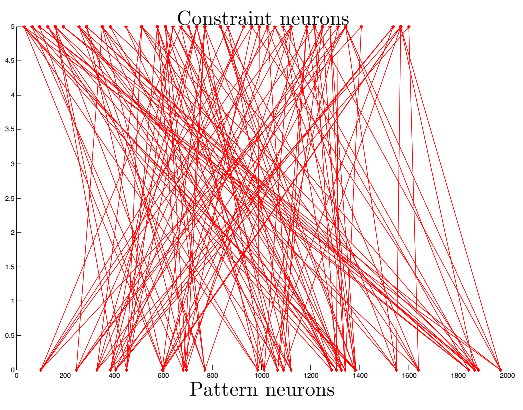
Theorem 43 only indicates that noisy neural networks (under our model) outperform noiseless ones. However, it does not specify the level of errors that such networks can correct. In the following, we derive a theoretical lower bound on the error correction performance of the recall algorithm. To this end, let P_{c_i} denote the average probability that a cluster can correct i external errors in its domain. The following theorem gives a simple condition under which Algorithm 14 can correct a linear fraction of external errors (in terms of n) with high probability. The condition involves $\widetilde{\lambda}$ and $\widetilde{\rho}$, the degree distributions of the contracted graph $\widetilde{G}$. As usual, the theorem holds if the decision subgraphs for the pattern neurons in graph $\widetilde{G}$ are tree-like for a depth of τL, where τ is the total number of number of iterations performed by Algorithm 14.

Theorem 44. *Under the assumptions that graph $\widetilde{G}$ grows large and it is chosen randomly with degree distributions given by $\widetilde{\lambda}$ and $\widetilde{\rho}$, Algorithm 14 is successful if*

$$p_e\widetilde{\lambda}\left(1 - \sum_{i\geq 1} P_{c_i}\frac{z^{i-1}}{i!} \cdot \frac{d^{i-1}\widetilde{\rho}(1-z)}{dz^{i-1}}\right) < z, \;\; for\ z \in [0, p_e]. \tag{10.6}$$



(a) $t = 0$



(b) $t \to \infty$

Figure 10.3: An external noise pattern that contains a stopping set in a noiseless neural circuit. Left panel shows the original pattern and the right one illustrates the result of the decoding algorithm after a sufficient number of iterations where the algorithm gets stuck. Blue pattern nodes are those that are connected to at least one cluster with a single external error, and thus, can be corrected during the recall phase. Obviously, the stopping set on the right does not have any blue nodes.

Proof. The proof is based on the density evolution technique [54]. Without loss of generality, assume that we have $P_{c_1} > 0$, $P_{c_2} > 0$ and $P_{c_3} > 0$ (and $P_{c_i} = 0$ for $i > 3$.) but the approach can easily be extended if we have $P_{c_i} \neq 0$ for $i > 3$. Let $\Pi(t)$ be the average probability that a super constraint node sends a failure message, i.e., that it can not correct external errors lying in its domain. Then, the probability that a noisy pattern neuron with degree d_i sends an erroneous message to a particular neighbor among super constraint node is equal to the probability that none of its *other* neighboring super constraint nodes could have corrected its error, i.e.,

$$P_i(t) = p_e(\Pi(t))^{d_i - 1}.$$

Averaging over d_i we find the average probability of error in iteration t:

$$z(t+1) = p_e\widetilde{\lambda}(\Pi(t)). \tag{10.7}$$

Now consider a cluster ℓ that contains d_ℓ pattern neurons. This cluster will *not* send a failure message to a noisy neighboring pattern neuron with probability:

1. P_{c_1}, if it is not connected to any other noisy neuron.
2. P_{c_2}, if it is connected to exactly one other constraint neuron.

3. P_{c_3}, if it is connected to exactly two other constraint neurons.

4. 0, if it is connected to more than two other constraint neuron.

Thus, we obtain

$$\begin{aligned}\Pi^{(\ell)}(t) &= 1 - P_{c_1}(1-z(t))^{d_\ell - 1} - P_{c_2}\binom{d_\ell - 1}{1} z(t)(1-z(t))^{d_\ell - 2} \\ &- P_{c_3}\binom{d_\ell - 1}{2} z(t)^2(1-z(t))^{d_\ell - 3}.\end{aligned}$$

Averaging over d_ℓ we obtain

$$\begin{aligned}\Pi(t) &= \mathbb{E}_{d_\ell}\left(\Pi^{(\ell)}(t)\right) = 1 - P_{c_1}\widetilde{\rho}(1-z(t)) - P_{c_2} z(t)\widetilde{\rho}'(1-z(t)) \\ &- 0.5 P_{c_2} z(t)^2 \widetilde{\rho}''(1-z(t)),\end{aligned} \tag{10.8}$$

where $\widetilde{\rho}'(z)$ and $\widetilde{\rho}''(z)$ are derivatives of the function $\widetilde{\rho}(z)$ with respect to z.

Equations (10.7) and (10.8) will give us the value of $z(t+1)$ as a function of $z(t)$. We can then calculate the final symbol error rate as $z(\infty)$. For $z(\infty) \to 0$, it is sufficient to have $z(t+1) < z(t)$, which proves the theorem.

□

Remark 45. *It must be noted that since the size of the neural graphs in this thesis are fairly limited, in some cases the decision subgraphs might not be tree-like. In those cases the performance might deviate from the one predicted by Theorem 44. Our simulation results, however, show that in many cases the proposed analytical approximation is a good estimate of the performance in the recall phase.*

Overall, it must be emphasized that the theoretical results derived in this chapter show that if *the neural connectivity matrix has certain properties then the recall algorithm could correct a linear number of erroneous symbols in the input pattern during the recall phase.*

Theorem 44 states that for any fraction of errors $\Lambda_{\upsilon,\nu} \leq \Lambda^*_{\upsilon,\nu}$ that satisfies the above recursive formula, Algorithm 14 will be successful with probability close to one. Note that the first fixed point of the above recursive equation dictates the maximum fraction of errors $\Lambda^*_{\upsilon,\nu}$ that our model can correct. For the special case of $P_{c_1} = 1$ and $P_{c_i} = 0, \forall i > 1$, we obtain $p_e\widetilde{\lambda}(1-\widetilde{\rho}(1-z)) < z$, the same condition given in Chapter 7. Theorem 44 takes into account the contribution of all P_{c_i} terms and, as we will shortly see, the maximum value of P_{c_i} does not occur when the internal noise is equal to zero, i.e., $\upsilon = \nu = 0$, but instead when the neurons are contaminated with internal noise! This also suggests that even individual clusters are able to correct more errors in the presence of internal noise.

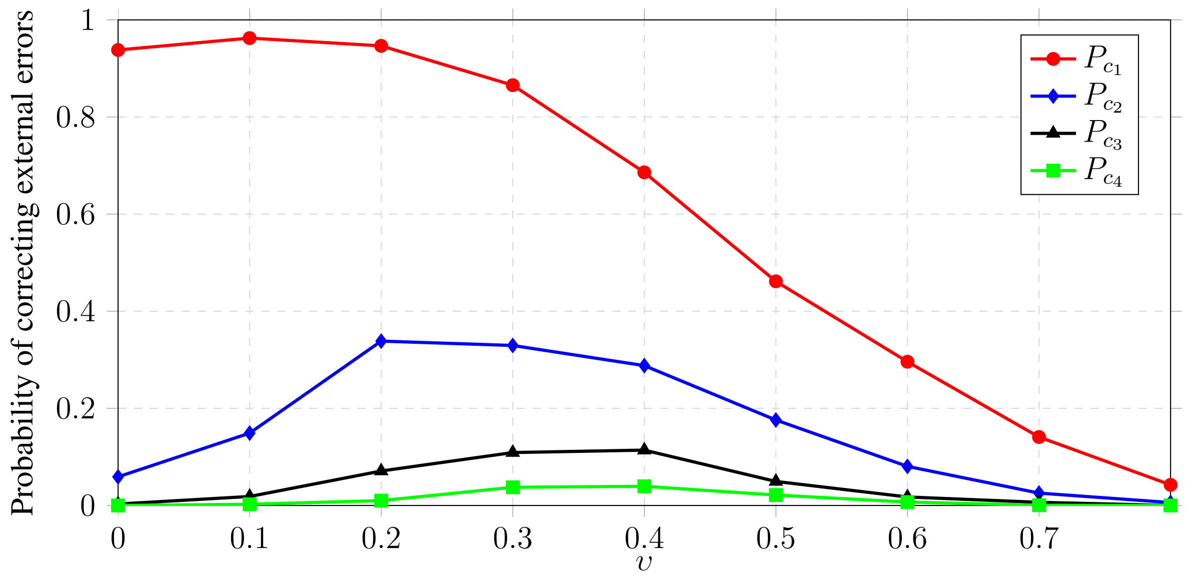


Figure 10.4: The value of P_{c_i} as a function of pattern neurons noise υ for $i = 1, \dots, 4$. The noise at constraint neurons is assumed to be zero ($\nu = 0$).

To estimate P_{c_i}'s, we use numerical approaches[6]. More specifically, given a set of clusters $W^{(1)}, \dots, W^{(L)}$, for each cluster we randomly corrupt i pattern neurons with ± 1 noise. Then, we let Algorithm 13 run over this cluster and calculate the success rate once finished. We take the average of this rate over all clusters to end up with P_{c_i}. The results of this approach are shown in Figure 10.4, where the value of P_{c_i} is shown for $i = 1, \dots, 4$ and various noise amounts at the pattern neurons (specified by parameter υ).

A typical trend of P_{c_i}'s in terms of ν is shown in Figure 10.4 (note that maximum values are not at $\upsilon = 0$). As mentioned earlier, we see that the maximum of P_{c_i} does not occur when the internal noise is equal to zero, i.e., $\upsilon = 0$ but when the pattern neurons are contaminated with internal noise! Thus, a fair amount of internal noise actually helps the network.

10.3 Simulations

We now consider the simulation results for a finite system. In order to learn the subspace constraints for each cluster $G^{(\ell)}$ we use the (learning) Algorithm 1, proposed in Chapter 5.

[6]We also provide an analytical approache to estimate P_{c_1} in the appendix. However, this technique involves lots of approximations, which makes the obtained bounds on P_{c_i}'s quite loose.

Henceforth, we assume that the weight matrix W is known and given. Thus, we will use the same network we learned in Chapter 7. In this setup, we consider a network of size $n = 400$ with $L = 50$ clusters. We have 40 pattern nodes and 20 constraint nodes in each cluster, on average. The external error is modeled by randomly generated vectors z where the entries are ± 1 with probability p_e and 0 otherwise. The vector z is added to the correct patterns that satisfy the orthogonality constraints. For denoising/recall, Algorithm 14 is used and the results are reported in terms of symbol error rate (SER) as we change the level of external error (p_e) or internal noise (υ, ν), i.e., counting the positions where the output of Algorithm 14 differs from the correct (noiseless) patterns.[7]

Symbol Error Rate as a function of Internal Noise

Figure 10.5 illustrates the final Symbol Error Rate (SER) of our proposed algorithm for different values of υ and ν. Recall that υ and ν quantify the level of noise in pattern and constraint neurons, respectively. Dashed lines in Figure 10.5 are simulation results whereas solid lines are theoretical upper bounds provided in this chapter. As evident from this figure, there is a threshold phenomenon such that SER is negligible for $p_e \leq p_e^*$ and grows beyond this threshold. As expected, simulation results are better than the theoretical bounds. In particular, the gap is relatively large as υ moves towards one.

A more interesting trend in Figure 10.5 is the fact internal noise helps in achieving better performance, as predicted by theoretical analysis (Theorem 43). Notice how p_e^* grows as ν increases.

This phenomenon is inspected more closely in Figure 10.6 where p_e is fixed to 0.125 while υ and ν vary. Figures 10.7a and 10.7b display projected versions of the surface plot to investigate the effect of υ and ν separately. As we see again, a moderate amount of internal noise at both pattern and constraint neurons improves performance. There is an optimum point (υ^*, ν^*) for which the SER reaches its minimum. Figure 10.7b indicates for instance that $\nu^* \approx 0.25$, beyond which SER deteriorates.

Recall Time as a function of Internal Noise

Figure 10.8 illustrates the number of iterations performed by Algorithm 14 for correcting the external errors when p_e is fixed to 0.075. We stop whenever the algorithm corrects all external errors or declare a recall error if all errors were not corrected in 40 iterations. Thus, the corresponding areas in the figure where the number of iterations reaches 40 indicates decoding failure. Figures 10.9a and 10.9b are projected versions of Figure 10.8 and show the average number of iterations as a function of υ and ν, respectively.

[7]The MATLAB code that is used in conducting the simulations mentioned in this chapter is available online at `https://github.com/saloot/NeuralAssociativeMemory`.

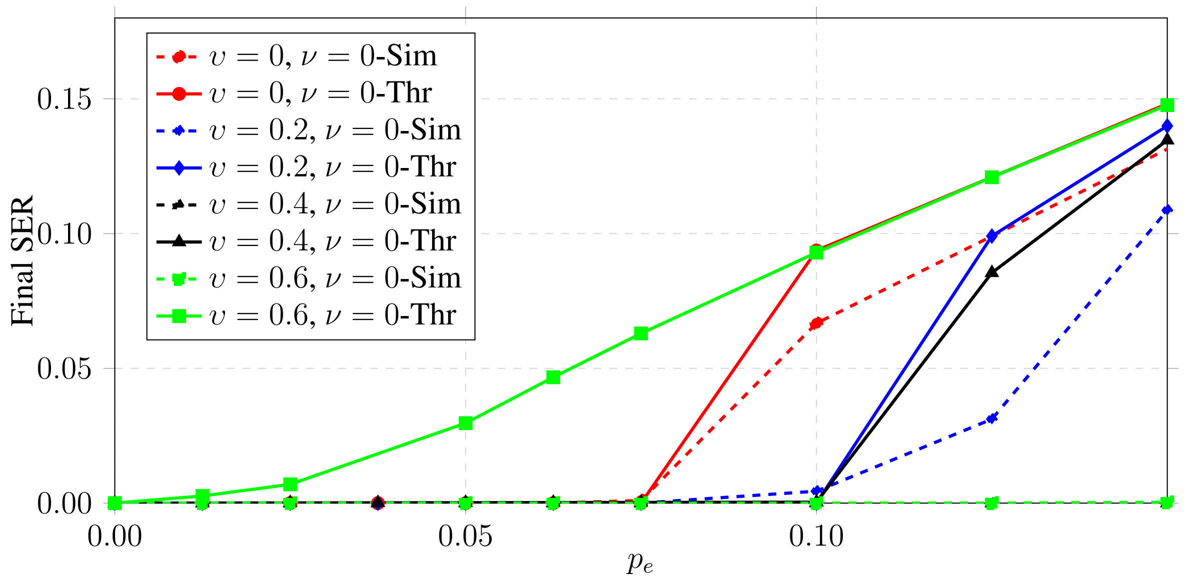


Figure 10.5: The final SER for a network with $n = 400$, $L = 50$ cf. Chapter 7. The red curves correspond to the noiseless neural network.

The amount of internal noise drastically affects the speed of Algorithm 14. First, from Figure 10.8 and 10.9a observe the running time is more sensitive to noise at pattern neurons than at constraint neurons. Furthermore, in the low noise regime at pattern neurons, the algorithms become slower as noise at constraint neurons is increased. Finally, increasing the noise at the pattern neurons (upto a certain limit) improves the running time, as seen in Figure 10.9a.

Note that the results presented here are for the case where the noiseless decoder succeeds as well and its average number of iterations is pretty close to the optimal value (see Figure 10.8). Figure 10.10 illustrates the number of iterations performed by Algorithm 14 for correcting the external errors when p_e is fixed to 0.125. In this case, the noiseless decoder encounters stopping sets while the noisy decoder is still capable of correcting external errors. Here we see that the optimal running time occurs when the neurons have a fair amount of internal noise. Figures 10.11a and 10.11b are projected versions of Figure 10.10 and show the average number of iterations as a function of υ and ν, respectively.

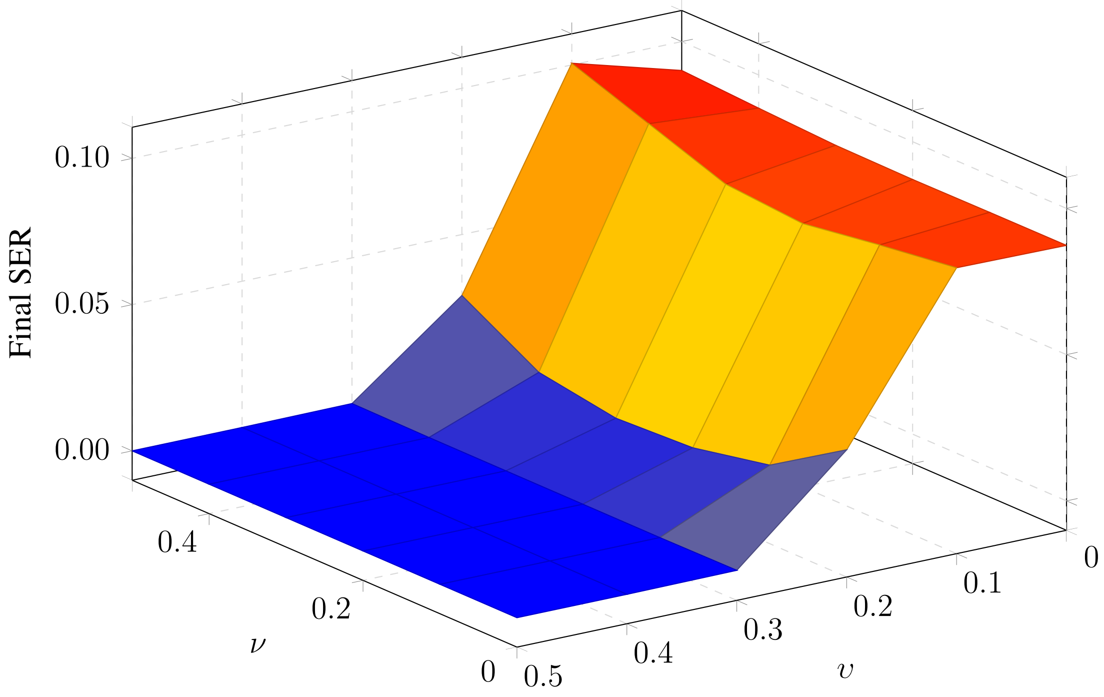


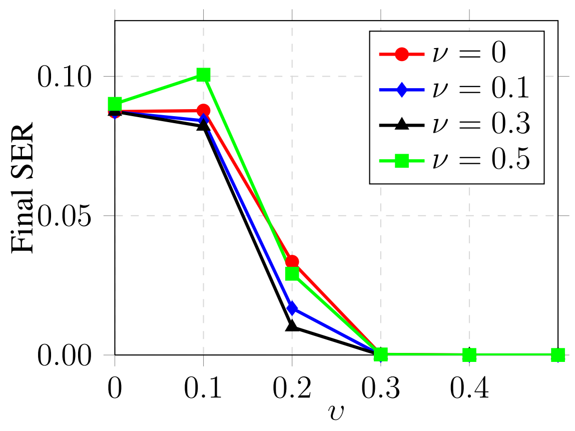
Figure 10.6: The final symbol error probability when $p_e = 0.125$ as a function of internal noise parameters at the pattern and constraint neurons, denoted by υ and ν, respectively.

Larger noise values

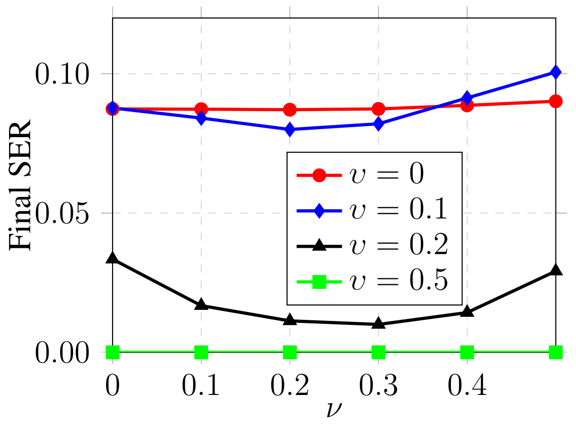
So far, we have investigated the performance of the recall algorithm when noise were limited to ± 1. Although this choice facilitates the analysis of the algorithm and increases error correction speed, our analysis is virtually valid for larger noise values. Figure 10.12 illustrates the Symbol Error Rate (SER) for the same scenario as before but with noise values chosen from $\{-3, -2, \dots, 2, 3\}$. We see exactly the same behavior as we witnessed for ± 1 noise values.

Effect of internal noise on the performance of the neural network in absence of external noise

Now we provide results of a study for a slightly modified scenario where there is only internal noise and no external errors and $\varphi < \upsilon$. Thus, the internal noise can now cause neurons to make wrong decisions, even in the absence of external errors. By abuse of notation, we assume the pattern neurons to be corrupted with a ± 1 noise added to them with probability



(a) Final SER as function of υ for $p_e = 0.125$.



(b) The effect of ν on the final SER for $p_e = 0.125$

Figure 10.7: The final symbol error probability for as a function of internal noise parameters at pattern and constraint neurons for $p_e = 0.125$

υ. The rest of the scenario is the same as before.

Figure 10.13 illustrates the effect of the internal noise as a function of υ and ν, the noise parameters at the pattern and constraint nodes, respectively. This behavior is shown in Figures 10.14a and 10.14b for better inspection.

Here, we witness the more familiar phenomenon where increasing the amount of internal noise results in a worse performance. This finding emphasizes the importance of choosing update threshold φ and ψ properly, according to Lemma 42.

Interestingly, this behavior is very similar to the effect of heat stress on the performance of wireless telegraphy operators [70, Figure 2]. The two phenomena might be related because external heat will translate into neurons with more internal thermal noise.

10.4 Final Remarks

We have demonstrated that associative memories still work reliably even when built from unreliable hardware, addressing a major problem in fault-tolerant computing and further arguing for the viability of associative memory models for the (noisy) mammalian brain. After all, brain regions modeled as associative memories, such as the hippocampus and the olfactory cortex, certainly do display internal noise [71–73]. Further, we found a threshold

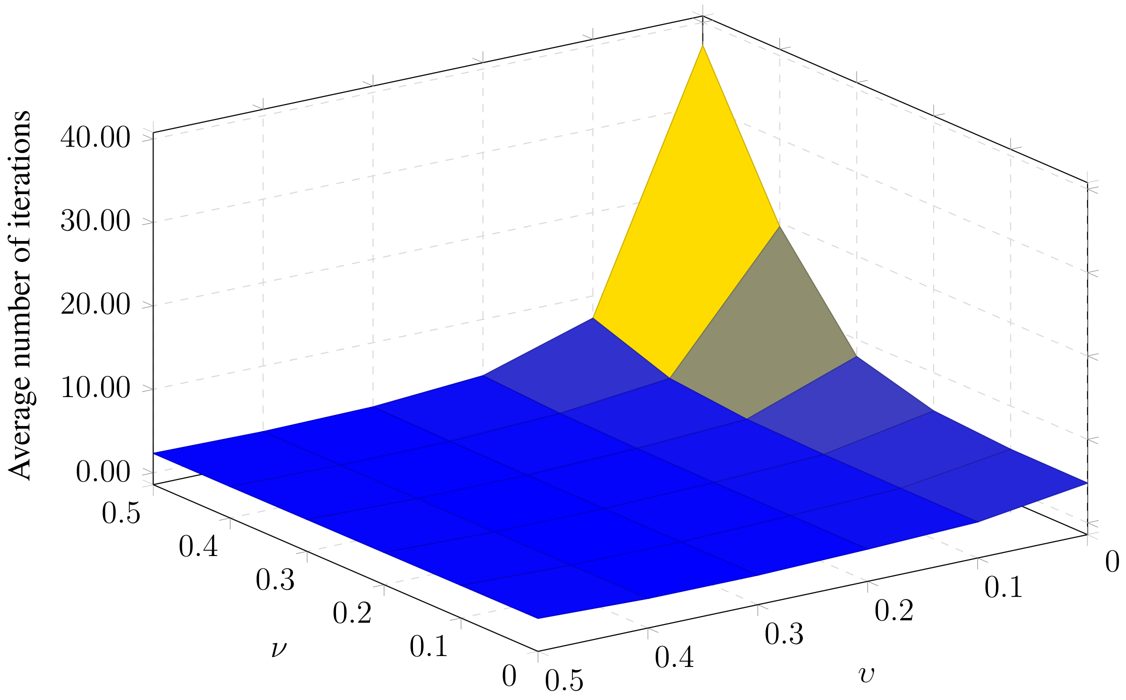
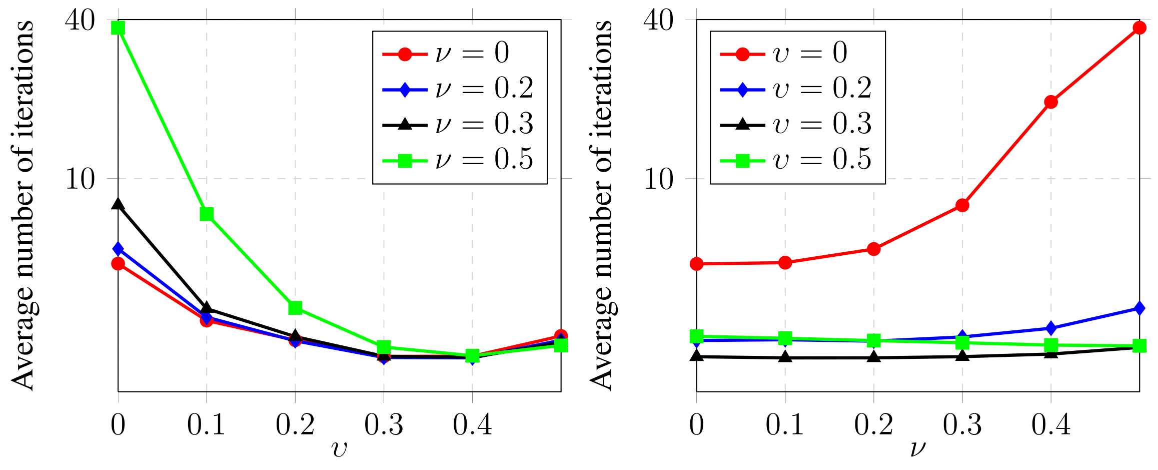


Figure 10.8: The effect of internal noise on the number of iterations performed by Algorithm 14, for different values of υ and ν with $p_e = 0.075$.

phenomenon for reliable operation in our model, which manifests the tradeoff between the amount of internal noise and the amount of external noise that the system can handle.

In fact, we showed that internal noise actually improves the performance of the network in dealing with external errors, up to some optimal value. This is a manifestation of the *stochastic facilitation* [73] or *noise enhancement* [74] phenomenon that has been observed in other neuronal and signal processing systems, providing a functional benefit to variability in the operation of neural systems.

The associative memory design developed herein uses thresholding operations in the message-passing algorithm for recall; as part of our investigation, we optimized these neural firing thresholds based on the statistics of the internal noise. As noted by Sarpeshkar in describing the properties of analog and digital computing circuits, "In a cascade of analog stages, noise starts to accumulate. Thus, complex systems with many stages are difficult to build. [In digital systems] Round-off error does not accumulate significantly for many computations. Thus, complex systems with many stages are easy to build" [75]. One key to our result is capturing this benefit of digital processing (thresholding to prevent the build up of errors due to internal noise) as well as a modular architecture which allows us to correct a



(a) Effect of internal noise at pattern neurons side.

(b) Effect of internal noise at constraint neurons side.

Figure 10.9: The effect of internal noise on the number of iterations performed by Algorithm 14, for different values of υ and ν with $p_e = 0.075$. The average iteration number of 40 indicate the failure of Algorithm 14.

linear number of external errors (in terms of the patterns length).

This chapter focused on recall, however learning is the other critical stage of associative memory operation. Indeed, information storage in nervous systems is said to be subject to storage (or learning) noise, *in situ* noise, and retrieval (or recall) noise [76, Fig. 1]. It should be noted, however, that there is no essential loss by combining learning noise and *in situ* noise into what we have called external error herein, cf. [69, Fn. 1 and Prop. 1]. Thus our basic qualitative result extends to the setting where the learning and stored phases are also performed with noisy hardware.

Going forward, it is of interest to investigate other neural information processing models that explicitly incorporate internal noise and to see whether they provide insight into observed empirical phenomena. As an example, we might be able to explain the threshold phenomenon observed in the symbol error rate of human telegraph operators under heat stress [70, Figure 2], by invoking a thermal internal noise explanation.

10.A Theoretical Estimation of P_{c_1}

To bound P_{c_1}, consider four event probabilities for a cluster, let us define

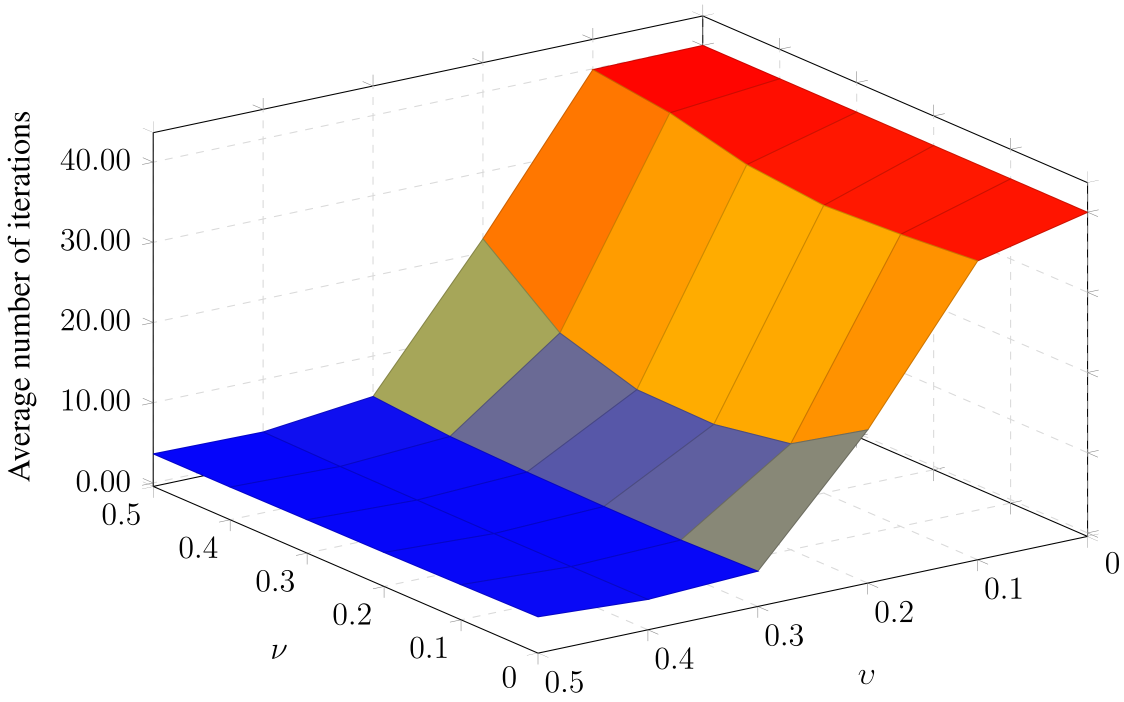


Figure 10.10: The effect of internal noise on the number of iterations performed by Algorithm 14, for different values of υ and ν with $p_e = 0.125$.

- $\pi_0^{(\ell)}$ (resp. $P_0^{(\ell)}$): The probability that a constraint neuron (resp. pattern neuron) in cluster ℓ makes a wrong decision due to its internal noise when there is no external noise introduced to cluster ℓ, i.e., $\|z^{(\ell)}\|_0 = 0$.

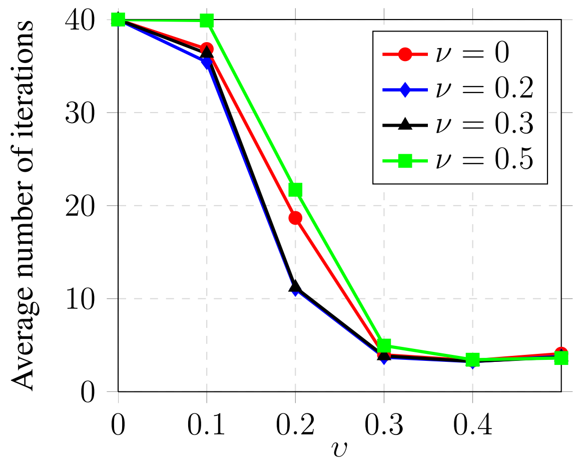
- $\pi_1^{(\ell)}$ (resp. $P_1^{(\ell)}$): The probability that a constraint neuron (resp. pattern neuron) in cluster ℓ makes a wrong decision due to its internal noise when one input error (external noise) is introduced, i.e., $\|z^{(\ell)}\|_0 = \|z^{(\ell)}\|_1 = 1$.

Notice that $P_{c_1}^{(\ell)} = 1 - P_1^{(\ell)}$.

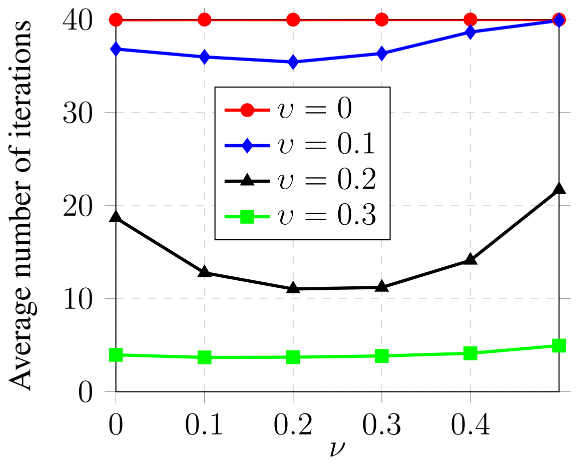
We derive an upper bound on the probability that a constraint node makes a mistake in the presence of one external error.

Lemma 46. *In the presence of a single external error, the probability that a constraint neuron in a given cluster ℓ makes a wrong decision due to its internal noise is given by*

$$\pi_1^{(\ell)} \leq \max\left(0, \frac{\nu - (\eta - \psi)}{2\nu}\right),$$



(a) Effect of internal noise at pattern neurons side.



(b) Effect of internal noise at constraint neurons side.

Figure 10.11: The effect of internal noise on the number of iterations performed by Algorithm 14, for different values of υ and ν with $p_e = 0.125$. The average iteration number of 40 indicate the failure of Algorithm 14.

where $\eta = \min_{i,j,W_{ij}^{(\ell)} \neq 0} \left(|W_{ij}^{(\ell)}| \right)$ *is the minimum absolute value of the non-zero weights in the neural graph and is chosen such that* $\eta \geq \psi$.[8]

Proof. Without loss of generality, we assume that it is the first pattern node, $x_1^{(\ell)}$, that is corrupted with noise whose value is $+1$. Now we would like to calculate the probability that a constraint node makes a mistake in such a circumstance. Furthermore, we will only consider the constraint neurons that are connected to $x_1^{(\ell)}$. Because for the other constraint neurons, the situation is the same as in the previous cases where there was no external noise (which we addressed in Lemma 42).

For a constraint neuron j that is connected to $x_1^{(\ell)}$, the decision parameter is

$$\begin{aligned} h_j^{(\ell)} &= \left(W^{(\ell)}.(x^{(\ell)} + z^{(\ell)}) \right)_j + v_j \\ &= 0 + \left(W^{(\ell)}.z^{(\ell)} \right)_j + v_j \\ &= w_{j1}^{(\ell)} + v_j. \end{aligned}$$

[8]This condition can be enforced during simulations as long as ψ is not too large, which itself is determined by the level of constraint neuron internal noise, ν, as we must have $\psi \geq \nu$.

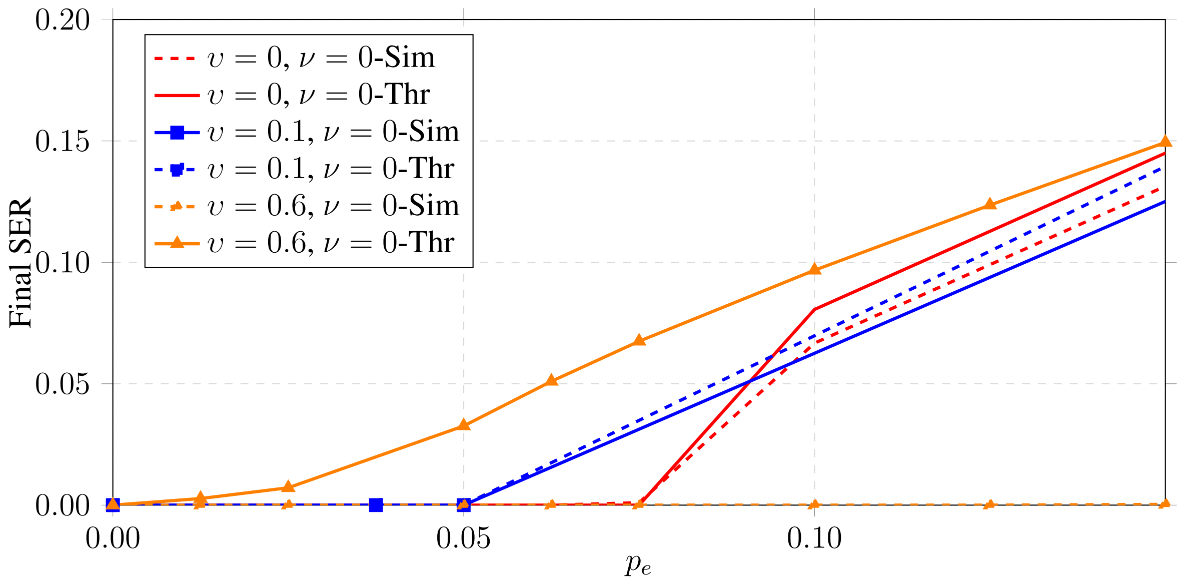


Figure 10.12: The final SER for a network with $n = 400$, $L = 50$ and noise values chosen from $\{-3, -2, \ldots, 2, 3\}$. The blue curves correspond to the noiseless neural network.

We consider two error events:

1. A constraint node j makes a mistake and does not send a message at all. The probability of this event is denoted by $\pi_1^{(\ell)}$.
2. A constraint node j makes a mistake and sends a message with the opposite sign. The probability of this event is denoted by $\pi_2^{(\ell)}$.

We first calculate the probability of $\pi_2^{(\ell)}$. Without loss of generality, assume $w_{j1}^{(\ell)} > 0$ so that the probability of an error of type two is as follows (the case for $w_{j1}^{(\ell)} < 0$ is exactly the same):

$$\begin{aligned} \pi_2^{(\ell)} &= \Pr\{w_{ji}^{(\ell)} + v_j < -\psi\} \\ &= \max\left(0, \frac{\nu - (\psi + w_{j1}^{(\ell)})}{2\nu}\right). \end{aligned} \tag{10.9}$$

However, since $\psi > \nu$ and $w_{j1}^{(\ell)} > 0$, then $\nu - (\psi + w_{j1}^{(\ell)}) < 0$ and $\pi_2^{(\ell)} = 0$. Therefore, the constraint neurons will never send a message that has an opposite sign to what it should have. All that remains to do is to calculate the probability that they remain silent by mistake.

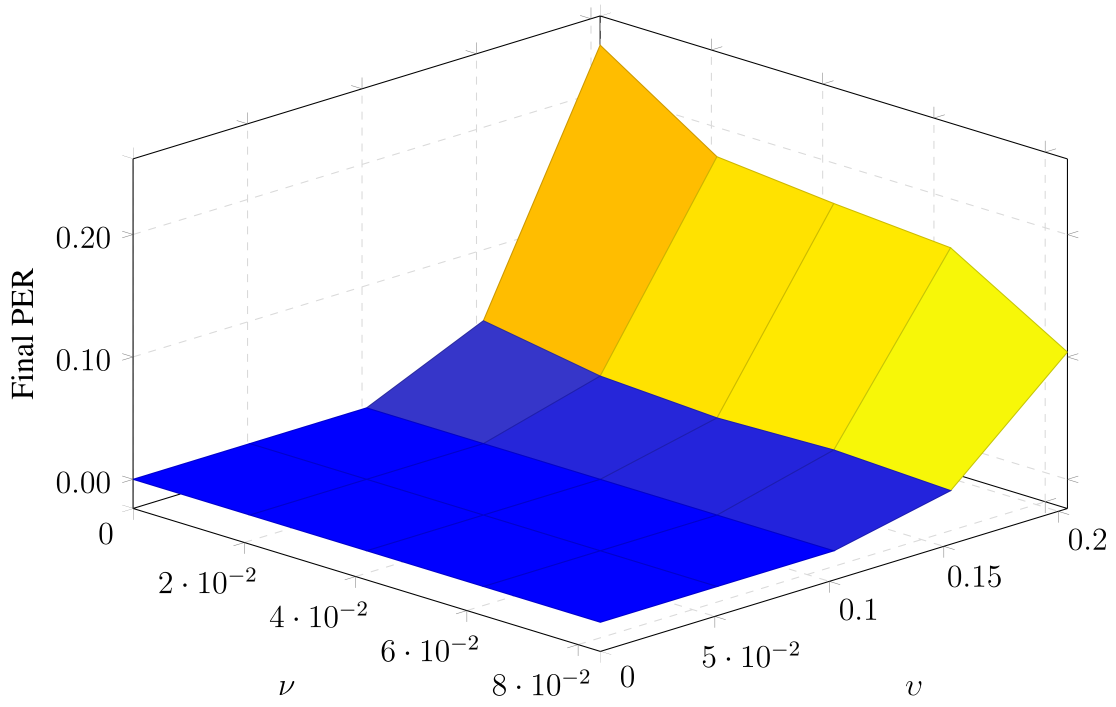


Figure 10.13: The effect of the internal noise on final PER as a function of υ and ν in absence of external noise.

To this end, we will have

$$\begin{aligned}\pi_1^{(\ell)} &= \Pr\{|w_{ji}^{(\ell)} + v_j| < \psi\} \\ &= \max\left(0, \frac{\nu + \min(\psi - w_{j1}^{(\ell)}, \nu)}{2\nu}\right). \end{aligned} \tag{10.10}$$

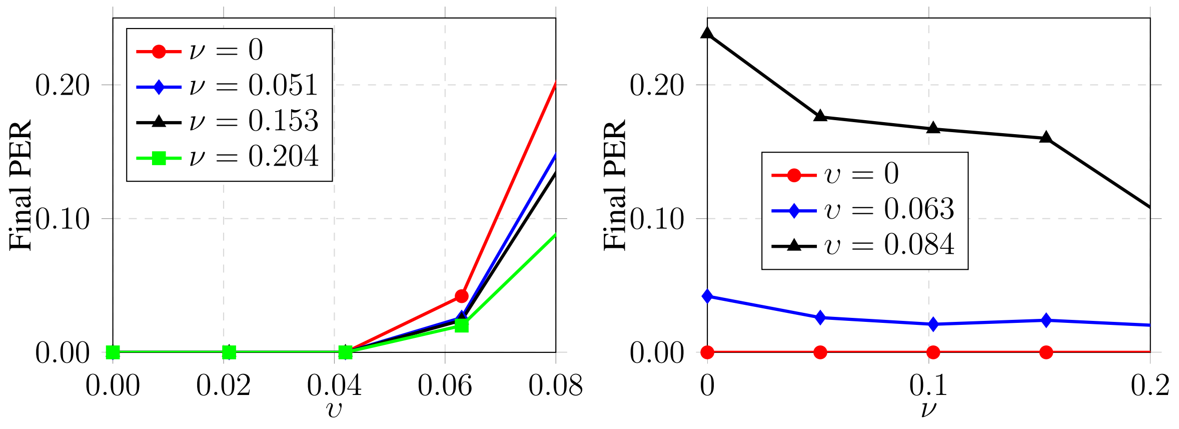
The above equation can be simplified if we assume that the absolute value of all weights in the network is bigger than a constant $\eta > \psi$. Then, the above equation will simplify to

$$\pi_1^{(\ell)} \leq \max\left(0, \frac{\nu - (\eta - \psi)}{2\nu}\right). \tag{10.11}$$

Putting the above equations together, we obtain:

$$\pi_{(1)} \leq \max\left(0, \frac{\nu - (\eta - \psi)}{2\nu}\right). \tag{10.12}$$

□



(a) Effect of internal noise at pattern neurons side.

(b) Effect of internal noise at constraint neurons side.

Figure 10.14: The effect of the internal noise on final PER as a function of υ and ν in absence of external noise.

In the case $\eta - \psi > \nu$, we can even manage to make this probability equal to zero. However, we will leave it as is and use (10.12) to calculate $P_1^{(\ell)}$.

Calculating $P_1^{(\ell)}$

We start by first calculating the probability that a non-corrupted pattern node $x_j^{(\ell)}$ makes a mistake, which is to change its state in round 1. Let us denote this probability by $q_1^{(\ell)}$. Now to calculate $q_1^{(\ell)}$ assume $x_j^{(\ell)}$ has degree d_j and it has b common neighbors with $x_1^{(\ell)}$, the corrupted pattern node.

Out of these b common neighbors, b_c will send ± 1 messages and the others will, mistakenly, send nothing. Thus, the decision making parameter of pattern node j, $g_j^{(\ell)}$, will be bounded by

$$g_j^{(\ell)} = \frac{\left(\text{sign}(W^{(\ell)})^\top \cdot y^{(\ell)}\right)_j}{d_j} + u_j. \leq \frac{b_c}{d_j} + u_j.$$

We will denote $\left(\text{sign}(W^{(\ell)})^\top \cdot y^{(\ell)}\right)_j$ by o_j for brevity from this point on.

In this circumstance, a mistake happens when $|g_j^{(\ell)}| \geq \varphi$. Thus

$$\begin{aligned} q_1^{(\ell)} &= \Pr\{|g_j^{(\ell)}| \geq \varphi | \deg(x_j^{(\ell)}) = d_j \ \& \ |\mathcal{N}(x_1^{(\ell)}) \cap \mathcal{N}(x_j^{(\ell)})| = b\} \\ &= \Pr\{\frac{o_j}{d_j} + u_j \geq \varphi\} + \Pr\{\frac{o_j}{d_j} + u_j \leq -\varphi\}, \end{aligned} \tag{10.13}$$

where $\mathcal{N}(x_i^{(\ell)})$ represents the neighborhood of pattern node $x_i^{(\ell)}$ among constraint nodes.

By simplifying (10.13) we obtain

$$q_1^{(\ell)}(o_j) = \begin{cases} 1, & \text{if } |o_j| \geq (\upsilon + \varphi) d_j, \\ \max(0, \frac{\upsilon - \varphi}{\upsilon}), & \text{if } |o_j| \leq |\upsilon - \varphi| d_j, \\ \frac{\upsilon - (\varphi - o_j/d_j)}{2\upsilon}, & \text{if } |o_j - \varphi d_j| \leq \upsilon d_j, \\ \frac{\upsilon - (\varphi + o_j/d_j)}{2\upsilon}, & \text{if } |o_j + \varphi d_j| \leq \upsilon d_j. \end{cases}$$

We now average this equation over o_j, b_c, b and d_j. To start, suppose out of the b_c non-zero messages the node $x_j^{(\ell)}$ receives, e of them have the same sign as the link they are being transmitted over. Thus, we will have $o_j = e - (b_c - e) = 2e - b_c$. Assuming the probability of having the same sign for each message is $1/2$, the probability of having e equal signs out of b_c elements will be $\binom{b_c}{e}(1/2)^{b_c}$. Thus, we will obtain

$$\bar{q}_1^{(\ell)} = \sum_{e=0}^{b_c} \binom{b_c}{e} (1/2)^{b_c} q_1^{(\ell)}(2e - b_c). \tag{10.14}$$

Now note that the probability of having $a - b_c$ mistakes from the constraint side is given by $\binom{b}{b_c}(\pi_1^{(\ell)})^{b-b_c}(1-\pi_1^{(\ell)})^{b_c}$. With some abuse of notation we obtain:

$$\bar{q}_1^{(\ell)} = \sum_{b_c=0}^{b} \binom{b}{b_c} (\pi_1^{(\ell)})^{b-b_c} (1 - \pi_1^{(\ell)})^{b_c} \sum_{e=0}^{b_c} \binom{b_c}{e} (1/2)^{b_c} q_1^{(\ell)}(2e - b_c). \tag{10.15}$$

Finally, the probability that $x_j^{(\ell)}$ and $x_1^{(\ell)}$ have b common neighbors can be approximated by $\binom{d_j}{b}(1 - \bar{d}^{(\ell)}/m_\ell)^{d_j - b}(\bar{d}^{(\ell)}/m_\ell)^b$, where $\bar{d}^{(\ell)}$ is the average degree of pattern nodes. Thus (again abusing some notation), we obtain:

$$\bar{q}_1^{(\ell)} \simeq \sum_{b=0}^{d_j} p_b \sum_{b_c=0}^{b} p_{b_c} \sum_{e=0}^{b_c} \binom{b_c}{e} (1/2)^{b_c} q_1^{(\ell)}(2e - b_c), \tag{10.16}$$

where $q_1^{(\ell)}(2e - b_c)$ is given by (10.13), p_b is the probability of having b common neighbors and is estimated by $\binom{d_j}{b}(1 - \bar{d}^{(\ell)}/m_\ell)^{d_j - b}(\bar{d}^{(\ell)}/m_\ell)^b$, with $\bar{d}^{(\ell)}$ being the average degree of pattern nodes in cluster ℓ. Furthermore, p_{b_c} is the probability of having $b - b_c$ out of these b nodes making mistakes. Hence, $p_{b_c} = \binom{b}{b_c}(\pi_1^{(\ell)})^{b-b_c}(1-\pi_1^{(\ell)})^{b_c}$. We will not simplify the above equation any further and use it as it is in our numerical analysis in order to obtain the best parameter φ.

Now we will turn our attention to the probability that the corrupted node, $x_1^{(\ell)}$, makes a mistake, which is either not to update at all or to update itself in the wrong direction. Recalling that we have assumed the external noise term in $x_1^{(\ell)}$ to be a $+1$ noise, the wrong direction would be for node $x_1^{(\ell)}$ to increase its current value instead of decreasing it. Furthermore, we assume that out of d_1 neighbors of $x_1^{(\ell)}$, some j of them have made a mistake and will not send any messages to $x_1^{(\ell)}$. Thus, the decision parameter of $x_1^{(\ell)}$ will be $g_1^{(\ell)} = u + (d_1 - j)/d_1$. Denoting the probability of making a mistake at $x_1^{(\ell)}$ by $q_2^{(\ell)}$ we will obtain

$$\begin{aligned} q_2^{(\ell)} &= \Pr\{g_1^{(\ell)} \leq \varphi |\ \deg(x_1^{(\ell)}) = d_1 \text{ and } j \text{ errors in constraints}\} \\ &= \Pr\{\frac{d_1 - j}{d_1} + u < \varphi\}, \end{aligned} \tag{10.17}$$

which simplifies to

$$q_2^{(\ell)}(j) = \begin{cases} +1, & \text{if } |j| \geq (1 + \upsilon - \varphi)d_1, \\ \max(0, \frac{\upsilon - \varphi}{\upsilon}), & \text{if } |j| \leq (1 - \upsilon - \varphi)d_1, \\ \frac{\upsilon + \varphi - (d_1 - j)/d_1}{2\upsilon}, & \text{if } |\varphi d_1 - (d_1 - j)| \leq \upsilon d_1. \end{cases} \tag{10.18}$$

Noting that the probability of making j mistakes on the constraint side is $\binom{d_1}{j}(\pi_1^{(\ell)})^j(1 - \pi_1^{(\ell)})^{d_1 - j}$, we get

$$\bar{q}_2^{(\ell)} = \sum_{j=0}^{d_1} \binom{d_1}{j} (\pi_1^{(\ell)})^j (1 - \pi_1^{(\ell)})^{d_1 - j} q_2^{(\ell)}(j), \tag{10.19}$$

where $q_2^{(\ell)}(j)$ is given by (10.18).

Putting the above results together, the overall probability of making a mistake on the side of pattern neurons when we have one bit of external noise is given by

$$P_1^{(\ell)} = \frac{1}{n^{(\ell)}} \bar{q}_2^{(\ell)} + \frac{n^{(\ell)} - 1}{n^{(\ell)}} \bar{q}_1^{(\ell)}. \tag{10.20}$$

Finally, the probability that cluster ℓ could correct one error is that all neurons take the correct decision, i.e.,

$$P_{c_1}^{(\ell)} = (1 - P_1^{(\ell)})^{n^{(\ell)}}$$

and the average probability that clusters could correct one error is simply

$$P_{c_1} = \mathbb{E}_\ell(P_{c_1}^{(\ell)}). \tag{10.21}$$

We will use this equation in order to find the best update threshold φ.

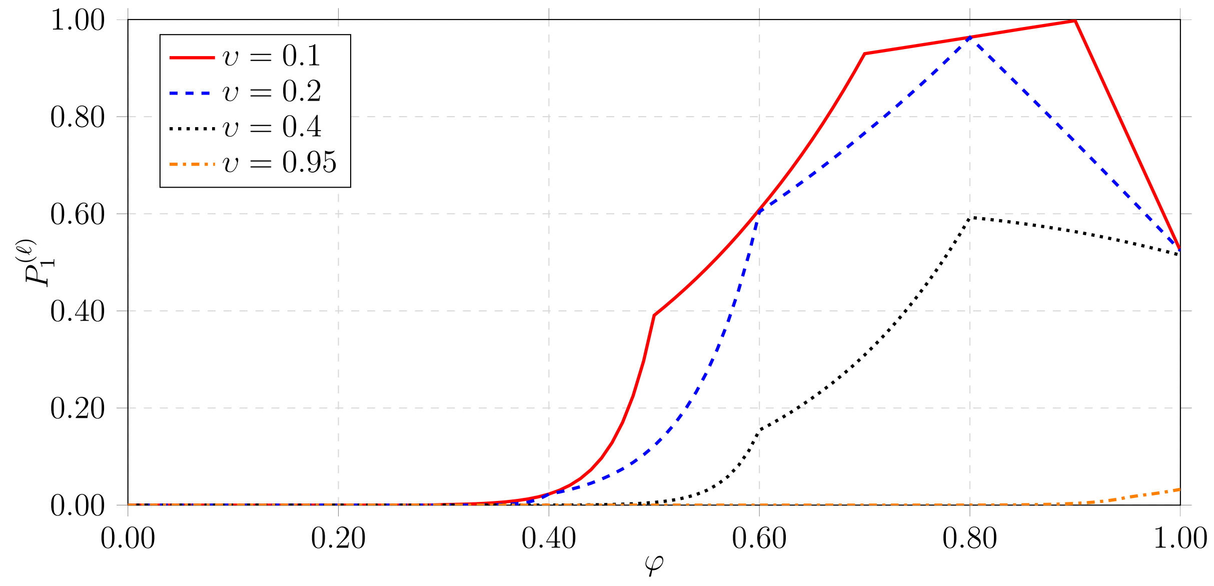


Figure 10.15: The behavior of P_{c_1} as a function of φ for different values of noise parameter, υ. Here, $\pi_{(1)} = 0.01$.

10.B Choosing a proper φ

We now apply numerical methods to (10.20) to find the best φ for different values of noise parameter υ. The following figures show the best choice for the parameter φ. The update threshold on the constraint side is chosen such that $\psi > \nu$. In each figure, we have illustrated the final probability of making a mistake, $P_1^{(\ell)}$, for comparison.

Figure 10.15 illustrates the behavior of the average probability of correcting a single error, P_{c_1}, as a function of φ for different values of υ and for $\pi_1 = 0.01$. The interesting trend here is that in all cases, φ^*, the update threshold that gives the best result, is chosen such that it is quite large. This actually is in line with our expectation because a small φ will result in non-corrupted nodes to update their states more frequently. On the other hand, a very large φ will prevent the corrupted nodes to correct their states, especially if there are some mistakes made on the constraint side, i.e., $\pi_1^{(\ell)} > 0$. Therefore, since we have much more non-corrupted nodes than corrupted nodes, it is best to choose a rather high φ but not too high. Please also note that when $\pi_1^{(\ell)}$ is very high, there are no values of υ for which error-free storage is possible.

Figure 10.16 illustrates the exact behavior of φ^* against υ for the case where $\phi_1 = 0$. As can be seen from the figure, φ should be quite large.

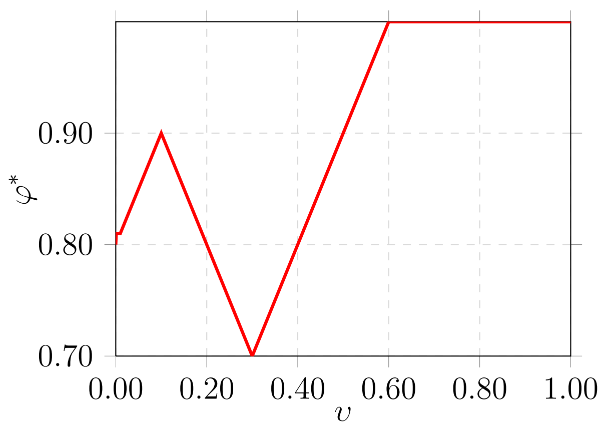


Figure 10.16: The behavior of φ^* as a function of υ for $\pi_1 = 0.01$.

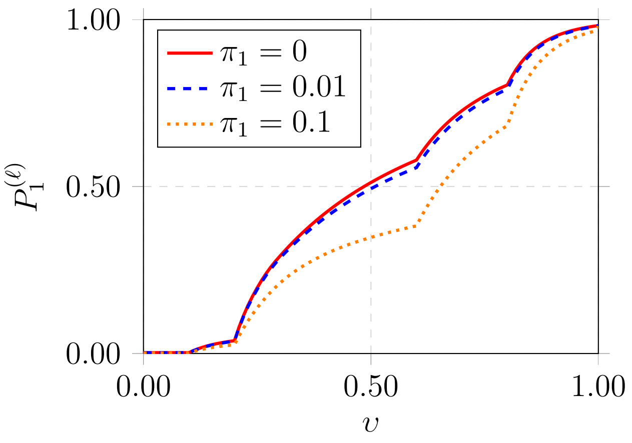


Figure 10.17: The optimum P_{e_1} as a function of υ for different values π_1.

Figure 10.17 illustrates $P_{e_1} = 1 - P_{c_1}$ for the best chosen threshold, φ^*, as a function of υ for various choices of π_1.

Chapter 11

Concluding Remarks and Open Problems

In this thesis our main goal was to show how one could combine ideas from coding theory with neural algorithms performing similar tasks in order to achieve much better efficiency in memorizing structured patterns. The proposed algorithms were designed to be simple enough to be implemented by a network of neurons and, yet, powerful to be able to handle partial information loss when recalling previously memorized patterns. We also showed how one could find a theoretical estimate on the performance of the proposed algorithms.

Nevertheless, there is still much to be done. Improving the performance of the suggested methods in practical situations aside, the most important issue that must be addressed in future is to use the proposed algorithms in memorizing datasets of natural patterns. The patterns in these datasets, despite being strongly correlated, often do not belong to a subspace. Consequently, one might not be able to directly apply the methods proposed to learn subspaces to such datasets. Although, the nonlinear neural associative memory proposed in Chapter 9 might provide some consolation to this limitation, it may turn out to be a bit too complex for many datasets of natural patterns.

More specifically, although the patterns in such datasets do not form a subspace, they come very close as they are strongly correlated and the correlation matrix of the patterns has a few large eigenvalues and the rest are very small (but not equal to zero). A good case in point is dataset of natural images. Figure 11.1 illustrates the eigenvalues for the correlation matrix of a dataset of 10000 images, uniformly sampled from 10 classes of the CIFAR-10 dataset [77]. As can be seen from the figure, there are few dominant principal

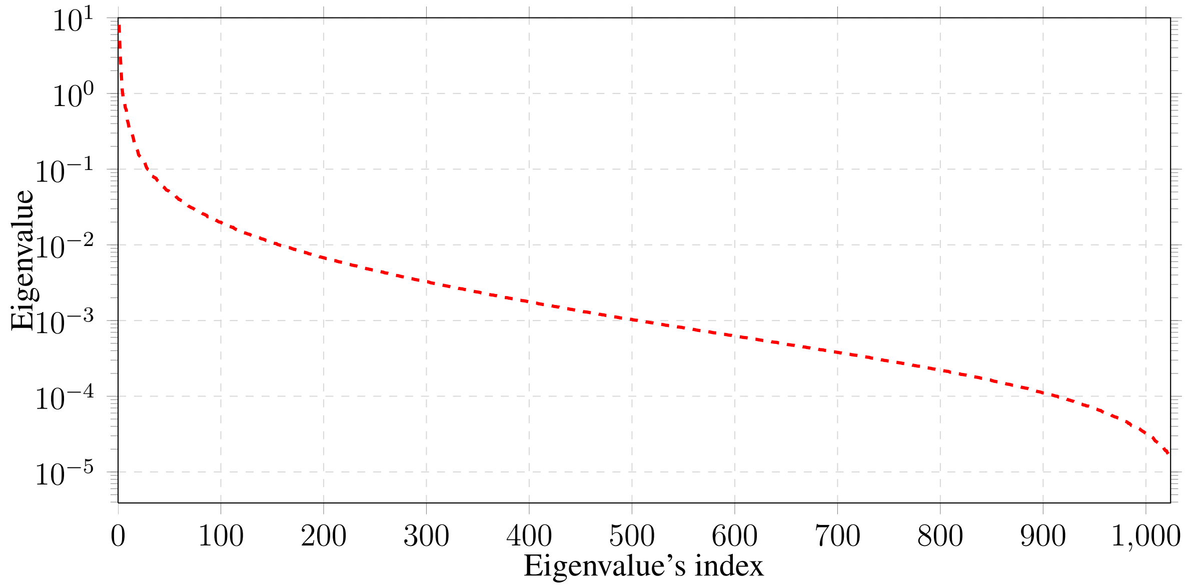
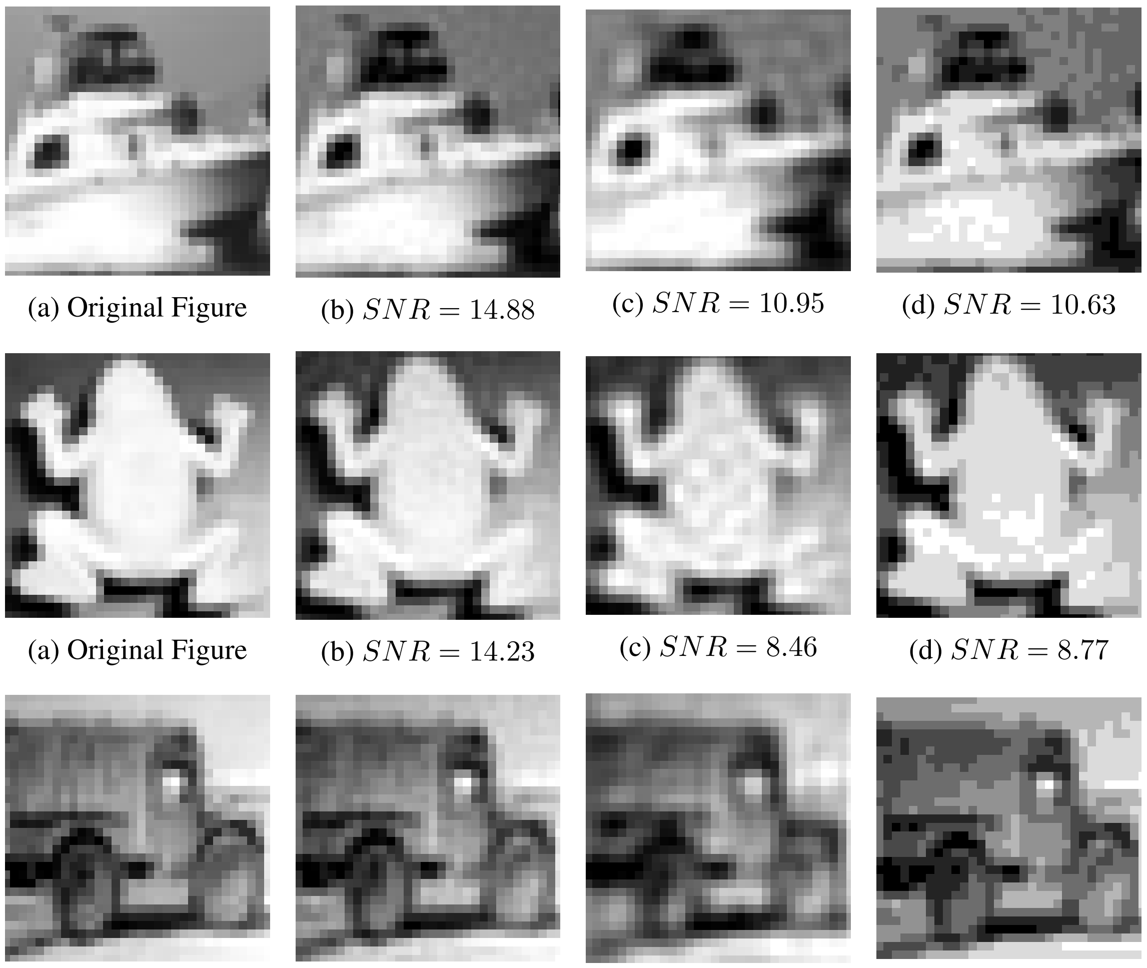


Figure 11.1: The eigenvalues of a dataset of 1000 32×32 gray-scale images, uniformly sampled from the 10 classes of the CIFAR-10 dataset [77].

components that can describe the patterns with a rather high accuracy, but there are non-zero minor components as well that prevent the patterns to form a subspace.

In this and similar cases, we might be able to slightly modify the previously proposed approaches in favor of practicality, i.e., instead of designing a method that can perfectly learn patterns that form a subspace, we focus on an approach that learns an *approximate* version of the given patterns, where the *important* aspects are preserved and only small details are sacrificed in order to accomplish the learning and the recall phases of neural associative memories. Think about it: if you are asked to describe how the sky looked like yesterday morning, you could easily describe it *qualitatively*, i.e., whether it was sunny or cloudy, blue or grey, and so on. However, remembering the exact details regarding the shape of the clouds might turn out to be an issue.

As a result, now what becomes important is to find a suitable *feature extraction* method that keep important details about the given patterns and focus on memorizing these features. A simple example could be to apply the Principal Components Analysis (PCA) and project the data onto its dominant principal components. We could then memorize the projected patterns without losing too much detail. Figure 11.2 illustrates some examples randomly drawn from the 10 classes of the CIFAR-10 dataset [77] and how they compare to their



(a) Original Figure (b) $SNR = 14.88$ (c) $SNR = 10.95$ (d) $SNR = 10.63$

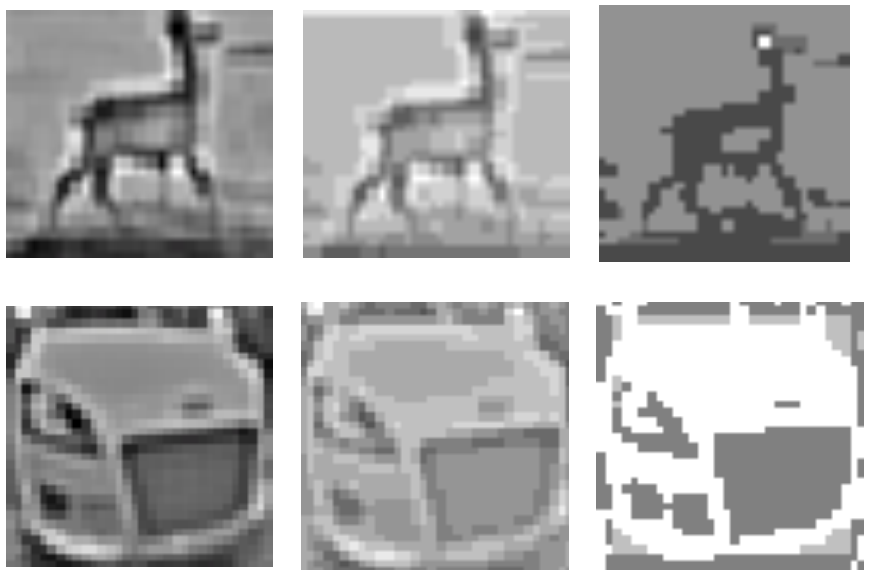
(a) Original Figure (b) $SNR = 14.23$ (c) $SNR = 8.46$ (d) $SNR = 8.77$

(a) Original Figure (b) $SNR = 11.37$ (c) $SNR = 6.95$ (d) $SNR = 7.29$

Figure 11.2: Panel (a) illustrates the original image. Panel (b) and (c) show the projected image over the 500 and 200 dominant Principal Components (PCs), respectively. Finally, panel (d) is the quantized version of images in panel (b) with 10 quantization levels. The *noise* in the SNR is calculated as the norm-2 of the difference between the original image and the image shown in the corresponding panel.

projected versions for different choices of dominant eigenvalues.

Currently, we are pursuing another direction with promising early results [78]. In the considered approach, we employ, for instance, the convolutional architecture to memorize gray-scale quantized images that are sampled from the CIFAR-10 dataset. However, al-



(a) Original image (b) Quantized image (c) Learned image

Figure 11.3: Original vs. learned images

though the images are close to forming a subspace, they do not exactly belong to one, as illustrated earlier in Figure 11.1. As such, we know that subspace learning algorithm will not work. More specifically, by the time the learning algorithm stops, we end up with a set of learned weights that are not orthogonal to the images but have minimal projections over the projection of the patterns (images).

At this point, it would be interesting to see what are the *fixed points* of the convolutional architecture with the neural graph determined by the learned vectors, i.e., the patterns that are close to the images in the dataset *and* orthogonal to the set of neural weights. These fixed points correspond to the patterns that are actually learned by the proposed architecture. Figure 11.3 illustrate two examples. Here, we see a very interesting trend: in many cases, the patterns learned by the network are very much similar to the actual image but with fewer details. It seems as if the network is focusing more on the important points within the images than the not as much important minute details.

Inspired by the results of Figure 11.3, if we replace the dataset of original images with those similar to the right column of Figure 11.3, the proposed architecture could learn all the images in this new dataset while being able to correct some errors during the recall phase as well. Based on these findings, it seems safe to suggest that in real life situations, in addition to the method used in designing associative memories, it is also as much important to think of suitable feature extraction algorithms, *tailored* for that particular associative memory scheme.

Extending this line of thought, and in light of recent developments in *deep* feature extracting architectures [57, 58, 60, 79, 80], it would be very interesting to think of multi-layer neural architectures that combine deep feature extraction and associative memory blocks. Such architectures could offer a complete neural associative memory package that works well for real patterns as well.

In addition, it would be worth mentioning that the convolutional model proposed in Chapter 7 is fairly general in the sense that as long as the small neural blocks correct a single error during the recall phase, all the theoretical and empirical investigations hold, even if the patterns do not belong to a subspace. In that regard, designing recall algorithms that could ensure correction of a single error within the (sub)patterns during the recall phase will also result in more general neural associative memories.

Bibliography

[1] H. Abdelbaki, E. Gelenbe, and S. E. El-Khamy, “Random neural network decoder for error correcting codes,” in *IEEE International Joint Conference on Neural Networks (IJCNN)*, vol. 5, 1999, pp. 3241–3245.

[2] A. Esposito, S. Rampone, and R. Tagliaferri, “A neural network for error correcting decoding of binary linear codes,” *Neural networks*, vol. 7, no. 1, pp. 195–202, 1994.

[3] A. G. Dimitrov and J. P. Miller, “Neural coding and decoding: communication channels and quantization,” *Network: Computation in Neural Systems*, vol. 12, no. 4, pp. 441–472, 2001.

[4] C. J. Rozell and D. H. Johnson, “Analyzing the robustness of redundant population codes in sensory and feature extraction systems,” *Neurocomputing*, vol. 69, no. 10, pp. 1215–1218, 2006.

[5] W. Gerstner and W. Kistler, *Spiking Neuron Models: An Introduction*. New York, NY, USA: Cambridge University Press, 2002.

[6] D. J. Amit, *Modeling brain function: The world of attractor neural networks*. New York, NY, USA: Cambridge University Press, 1992.

[7] K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (cam) circuits and architectures: A tutorial and survey,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[8] J. Chen, H. Guo, W. Wu, and W. Wang, “imecho: an associative memory based desktop search system,” in *ACM conference on Information and knowledge management*, 2009, pp. 731–740.

[9] C. Stangor, *Beginning Psychology*. Anonymous publisher, 2012.

[10] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA, USA: MIT Press, 2005.

[11] F. Rieke, D. Warland, R. de Ruyter van Steveninck, and W. Bialek, *Spikes: exploring the neural code*. Cambridge, MA, USA: MIT Press, 1999.

[12] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[13] J. Hertz, R. G. Palmer, and A. S. Krogh, *Introduction to the Theory of Neural Computation*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991.

[14] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 379, pp. 948–958, 1948.

[15] G. D. Forney Jr, "Concatenated codes." Ph.D. dissertation, Massachusetts Institute of Technology, 1965.

[16] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[17] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[18] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[19] R. Gallager, *Low-density parity-check codes*. Cambridge, MA, USA: MIT press, 1963.

[20] A. H. Salavati, K. R. Kumar, M. A. Shokrollahi, and W. Gerstner, "Neural pre-coding increases the pattern retrieval capacity of hopfield and bidirectional associative memories," in *IEEE International Symposium on Information Theory (ISIT)*, 2011, pp. 850–854.

[21] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 79, no. 8, pp. 2554–2558, 1982.

[22] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York, USA: Wiley & Sons, 1949.

[23] D. J. Amit, H. Gutfreund, and H. Sompolinsky, "Storing infinite numbers of patterns in a spin-glass model of neural networks," *Physical Review Letters*, vol. 55, pp. 1530–1533, 1985.

[24] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the hopfield associative memory," *IEEE Transactions on Information Theory*, vol. 33, no. 4, pp. 461–482, 1987.

[25] S. S. Venkatesh and D. Psaltis, "Linear and logarithmic capacities in associative neural networks," *IEEE Transactions on Information Theory*, vol. 35, no. 3, pp. 558–568, 1989.

[26] J. Komlós and R. Paturi, "Journal of computer and system sciences," *J. Comput. Syst. Sci.*, vol. 47, no. 2, pp. 350–373, 1993.

[27] M. Tsodyks and M. Feigel'Man, "The enhanced storage capacity in neural networks with low activity level," *EPL (Europhysics Letters)*, vol. 6, no. 2, p. 101, 1988.

[28] J. Buhmann, R. Divko, and K. Schulten, "Associative memory with high information content," *Physical Review A*, vol. 39, no. 5, p. 2689, 1989.

[29] C. Berrou and V. Gripon, "Coded Hopfield networks," in *IEEE International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, 2010, pp. 1–5.

[30] R. H. Gold, "Optimal binary sequences for spread spectrum multiplexing," *IEEE Transactions on Information Theory*, vol. 13, no. 4, pp. 619–621, 1967.

[31] T. Helleseth and P. V. Kumar, "Codes and sequences over $\mathbb{Z}_4$ - a tutorial overview," in *Difference Sets, Sequences and their Correlation Properties, NATO Science Series*. Kluwer Academic Publishers, 1999.

[32] J. G. Proakis, *Digital Communications*. New York, USA: McGraw-Hill International, 2001.

[33] L. Welch, "Lower bounds on the maximum cross correlation of signals (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 3, pp. 397–399, 2006.

[34] K. R. Kumar, A. H. Salavati, and M. A. Shokrollah, "Exponential pattern retrieval capacity with non-binary associative memory," in *IEEE Information Theory Workshop (ITW)*, 2011, pp. 80–84.

[35] K. R. Kumar, A. H. Salavati, and M. A. Shokrollahi, "A non-binary associative memory with exponential pattern retrieval capacity and iterative learning," *IEEE Transactions on Neural Networks and Learning systems*.

[36] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *Math. Analysis and Applications*, vol. 106, pp. 69–84, 1985.

[37] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 81, no. 10, pp. 3088–3092, 1984.

[38] S. Jankowski, A. Lozowski, and J. M. Zurada, "Complex-valued multistate neural associative memory." *IEEE Transactions on Neural Networks and Learning Systems*, vol. 7, no. 6, pp. 1491–1496, 1996.

[39] M. K. Muezzinoglu, C. Guzelis, and J. M. Zurada, “A new design method for the complex-valued multistate hopfield associative memory,” *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 891–899, 2003.

[40] D.-L. Lee, “Improvements of complex-valued Hopfield associative memory by using generalized projection rules.” *IEEE Transactions on Neural Networks*, vol. 17, no. 5, pp. 1341–1347, 2006.

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

[42] P. Peretto and J. J. Niez, “Long term memory storage capacity of multiconnected neural networks,” *Biological Cybernetics*, vol. 54, no. 1, pp. 53–64, 1986.

[43] L. Xu, A. Krzyzak, and E. Oja, “Neural nets for dual subspace pattern recognition method,” *International Journal of Neural Systems*, vol. 2, no. 3, pp. 169–184, 1991.

[44] E. Oja and T. Kohonen, “The subspace learning algorithm as a formalism for pattern recognition and neural networks,” in *IEEE International Conference on Neural Networks*, vol. 1, 1988, pp. 277–284.

[45] E. J. Candès and T. Tao, “Near-optimal signal recovery from random projections: Universal encoding strategies?” *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5406–5425, 2006.

[46] D. L. Donoho, A. Maleki, and A. Montanari, “Message-passing algorithms for compressed sensing,” *Proceedings of the National Academy of Sciences (PNAS)*, vol. 106, no. 45, pp. 18 914–18 919, 2009.

[47] J. A. Tropp and S. J. Wright, “Computational methods for sparse solution of linear inverse problems,” *Proceedings of the IEEE*, vol. 98, no. 6, pp. 948–958, 2010.

[48] A. H. Salavati and A. Karbasi, “Multi-level error-resilient neural networks,” in *IEEE International Symposium on Information Theory (ISIT)*, 2012, pp. 1064–1068.

[49] M. Sipser and D. A. Spielman, “Expander codes,” *IEEE Transactions on Information Theory*, vol. 42, pp. 1710–1722, 1996.

[50] S. Jafarpour, W. Xu, B. Hassibi, and A. R. Calderbank, “Efficient and robust compressed sensing using optimized expander graphs,” *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4299–4308, 2009.

[51] R. Motwani and P. Raghavan, *Randomized algorithms*. Cambridge university Press, 1995.

[52] D. Burshtein and G. Miller, “Expander graph arguments for message-passing algorithms,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 782–790, 2001.

[53] S. S. Venkatesh, "Connectivity versus capacity in the Hebb rule," in *Theoretical Advances in Neural Computation and Learning*. Springer, 1994, pp. 173–240.

[54] T. Richardson and R. Urbanke, *Modern Coding Theory*. New York, NY, USA: Cambridge University Press, 2008.

[55] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, 2001.

[56] A. Karbasi, A. H. Salavati, and A. Shokrollahi, "Iterative learning and denoising in convolutional neural associative memories," in *International Conference on Machine Learning (ICML)*, 2013, pp. 445–453.

[57] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *IEEE International Conference on Computer Vision (ICCV)*, 2009, pp. 2146–2153.

[58] Q. V. Le, J. Ngiam, Z. Chen, D. J. hao Chia, P. W. Koh, and A. Y. Ng, "Tiled convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2010, pp. 1279–1287.

[59] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 151–161.

[60] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *International Conference on Machine Learning (ICML)*, 2008, pp. 1096–1103.

[61] A. Yedla, Y.-Y. Jian, P. S. Nguyen, and H. D. Pfister, "A simple proof of threshold saturation for coupled vector recursions," in *IEEE International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, 2012, pp. 51–55.

[62] D. S. Modha, R. Ananthanarayanan, S. K. Esser, A. Ndirango, A. J. Sherbondy, and R. Singh, "Cognitive computing," *Communications of the ACM*, vol. 54, no. 8, pp. 62–71, 2011.

[63] A. Karbasi, A. H. Salavati, and A. Shokrollahi, "Coupled neural associative memories," in *IEEE Information Theory Workshop (ITW)*, 2013, pp. 1–5.

[64] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the bec," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 803–834, 2011.

[65] Y.-Y. Jian, H. D. Pfister, and K. R. Narayanan, “Approaching capacity at high rates with iterative hard-decision decoding,” in *IEEE International Symposium on Information Theory (ISIT)*, 2012, pp. 2696–2700.

[66] A. Karbasi, A. H. Salavati, A. Shokrollahi, and L. R. Varshney, “Noise-enhanced associative memorie,” in *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[67] M. G. Taylor, “Reliable information storage in memories designed from unreliable components,” *Bell System Technical Journal*, vol. 47, no. 10, pp. 2299–2337, 1968.

[68] A. V. Kuznetsov, “Information storage in a memory assembled from unreliable components,” *Problemy Peredachi Informatsii*, vol. 9, no. 3, pp. 100–114, 1973.

[69] L. R. Varshney, “Performance of LDPC codes under faulty iterative decoding,” *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, 2011.

[70] N. H. Mackworth, “Effects of heat on wireless telegraphy operators hearing and recording Morse messages,” *British Journal of Industrial Medicine.*, vol. 3, no. 3, pp. 143–158, 1946.

[71] C. Koch, *Biophysics of Computation: Information Processing in Single Neurons*. New York: Oxford University Press, 1999.

[72] M. Yoshida, H. Hayashi, K. Tateno, and S. Ishizuka, “Stochastic resonance in the hippocampal CA3–CA1 model: a possible memory recall mechanism,” *Neural Networks*, vol. 15, no. 10, pp. 1171–1183, 2002.

[73] M. D. McDonnell and L. M. Ward, “The benefits of noise in neural systems: bridging theory and experiment,” *Nature Reviews Neuroscience*, vol. 12, no. 7, pp. 415–426, 2011.

[74] H. Chen, P. K. Varshney, S. M. Kay, and J. H. Michels, “Theory of the stochastic resonance effect in signal detection: Part I–fixed detectors,” *IEEE Transactions on Signal Processing*, vol. 55, no. 7, pp. 3172–3184, 2007.

[75] R. Sarpeshkar, “Analog versus digital: Extrapolating from electronics to neurobiology,” *Neural Computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[76] L. R. Varshney, P. J. Sjöström, and D. B. Chklovskii, “Optimal information storage in noisy synapses under resource constraints,” *Neuron*, vol. 52, no. 3, pp. 409–423, 2006.

[77] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” *Master’s thesis, Department of Computer Science, University of Toronto*, 2009.

[78] A. Karbasi, A. H. Salavati, and A. Shokrollahi, “Convolutional neural associative memories,” in *To be submitted to the Journal of Machine Learning Research (JMLR)*, 2014.

[79] A. Coates and A. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *International Conference on Machine Learning (ICML)*, L. Getoor and T. Scheffer, Eds. New York, NY, USA: ACM, 2011, pp. 921–928.

[80] J. Ngiam, P. W. Koh, Z. Chen, S. A. Bhaskar, and A. Y. Ng, "Sparse filtering," in *Advances in Neural Information Processing Systems (NIPS)*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 1125–1133.

Amir Hesam Salavati

EPFL, IC, ALGO, BC 160 (Bâtiment BC), Station 14, CH-1015, Lausanne
+41216938137 • saloot@gmail.com • URL: http://algo.epfl.ch/~amir

Education

PhD in Computer and Communication Sciences

Ecole Polytechnique Federale de Lausanne (EPFL) **2009-2014**
Thesis: *Coding Theory and Neural Associative Memories with Exponential Pattern Retrieval Capacity*

M.Sc. in Communication Systems

Sharif University of Technology **2006-2008**
Thesis: *Quality of Service Network Coding*
Ranked 1st in communication systems students, Sharif U. of Tech., 2008

B.Sc. in Electrical Engineering

Sharif University of Technology **2002-2006**
Thesis: *Coded Pulse Anti-Clutter Systems (CPACS)*
Ranked 1st among nearly 450000 participants in the Iranian national entrance exam for universities (Concours), 2002

Research Interests

Neural networks, Coding Theory, Machine Learning, Graphical models

Publications

Neural Networks & Coding Theory

- A. H. Salavati, K.R. Kumar, A. Shokrollahi, W. Gerstner, "Neural Pre-coding Increases the Pattern Retrieval Capacity of Hopfield and Bidirectional Associative Memories", IEEE Int. Symp. Inf. Theory (ISIT), 2011.
- K. R. Kumar, A. H. Salavati, A. Shokrollahi, "Exponential Pattern Retrieval Capacity with Non-Binary Associative Memory", IEEE Inf. Theory Workshop (ITW), 2011.
- A. H. Salavati, A. Karbasi, "Multi-Level Error-Resilient Neural Networks", IEEE Int. Symp. Inf. Theory (ISIT), 2012.
- K. R. Kumar, A. H. Salavati, A. Shokrollahi, "Neural Associative Memory with Exponential Pattern Retrieval Capacity and Learning Abilities", IEEE Transactions on Neural Networks and Learning Systems, 2014.
- A. Karbasi, A. H. Salavati, A. Shokrollahi, "Iterative Learning and Denoising in Convolutional Neural Associative Memories", Int. Conf. Machine Learning (ICML), 2013.
- A. Karbasi, A. H. Salavati, A. Shokrollahi, "Coupled Neural Associative Memories", IEEE Inf. Theory Workshop (ITW), 2013.
- A. Karbasi, A. H. Salavati, A. Shokrollahi, L. R. Varshney, "Noise-Enhanced Associative Memories", Advances in Neural Inf. Processing Systems (NIPS), 2013.
- A. Karbasi, A. H. Salavati, A. Shokrollahi, "Convolutional Neural Associative Memories", *To be submitted to J. Machine Learning Research (JMLR), 2014.*
- A. Karbasi, A. H. Salavati, A. Shokrollahi, L. R. Varshney, "Noise Facilitation in Associative Memories of Exponential Capacity", *To be submitted to Neural Computation, 2014.*

Coding Theory

- K. Kumar, R. Kumar, P. Pakzad, A.H. Salavati, M.A. Shokrollahi, "Phase Transitions for Mutual Information", Proc. 6th Intl. Symposium on Turbo Codes and Iterative Information Processing (ISTC) 2010.

QOSNetwork Coding

- A. H. Salavati, B. H. Khalaj, M. R. Aref, “A Novel Approach for Providing QoS with Network Coding”, Proc. Int. Symp. Telecomm. (IST), 2008.

Bio-inspired Networking

- A. H. Salavati, H. Goudarzi, M. R. Pakravan, “An Ant Based Rate Allocation Algorithm For Media Streaming in Peer to Peer Networks”, Proc. Local Comp. Net. (LCN) 2008.
- H. Goudarzi, A. H. Salavati, M. R. Pakravan, “An ant-based rate allocation algorithm for media streaming in peer to peer networks: Extension to multiple sessions and dynamic networks”, J. Net. & Comp. App., 2011.

Research and Work Experience

Neuroscience and Bioinformatics

Research on Applications of Coding Theory in Neuroscience 2010-present

Working on applications of coding theory in analyzing the neural networks responsible for pattern recognition to increase their efficiency

Applications of Information Theory in Bioinformatics 2005 – 2009

Modeling genome as a written book to find its linguistic features and applying standard information theoretical algorithms in linguistics to analyze genome

Communication Networks

Research on Information Theory and Coding 2009 - 2011

Working on information theoretical aspects of fountain and LT codes.

Research on Quality of Service Network Coding 2006 - 2009

Working on novel algorithms to provide QoS using network coding.

Research on Bio-inspired Networking 2007 -2009

Working on applications of ant-inspired ideas in routing and rate allocation algorithms in media streaming P2P networks.

R&D Work in Datak Telecom. Summer 2006

Member of wireless R&D group at Datak Telecom, one of the pioneer ISPs in Iran

Robotics

Building a Maze Solver Micromouses 2004-2006

Building a micromouse able to find its way through an unknown maze autonomously. I was the head of motion controlling unit.

- 5th Place, Technical Innovation Award and 1st prize in new comers of 24th International Micromouse competition, Birmingham, UK, June 2005
- Ranked 3rd in the 1st national maze solver and line follower robotic competition, 2006, Iran

Some Academic Activities

- Vice president of EPFL IEEE student branch 2011–2012
- Treasurer of Iranian Students Association (IRSA) at EPFL 2011–2012
- Member of EPFL Graduate Student Association (GSA) 2009-2010
- President of Sharif University of Tech. IEEE student branch 2008–2009
- Vice president of Sharif University of Tech. IEEE student branch 2007–2008
- Chair of the scientific branch of Resana, Student’s extra curricular activities assoc. of Elect. Eng. Dep., Sharif U. of Tech. 2004-2005
- Chief Editor of Sharif university electrical engineering magazine 2004-2006
- Elected as a Member of students’ council at Sharif 2003-2004

Teaching Experience

Teaching assistant	Course: Algorithmique Instructor: Prof. M. A. Shokrollahi	Fall 2009, 2010, 2011
Teaching assistant	Course: Signal Processing for Communications, Instructor: Dr. P. Prandoni	Spring 2012
Teaching assistant	Course: Signals and Systems, Instructor: Dr. M. Babaie Zadeh	Spring 2008
Teaching assistant	Course: Communications Systems I, Instructor: Dr. M. R. Pakravan	Spring 2008

Training Courses

Cellular Networks

- "An Introduction to GSM", By Mobile Communications company of Iran (MCI), winter 2007 (passed the final exam with full mark)
- "Signaling in GSM", By Mobile Communications company of Iran (MCI), summer 2007

Web Development

- "Web Development", Taught by *Steve Huffman*, Offered by *Udacity*, 2013
- "Mobile Web Development", Taught by *Chris Wilson*, *Udacity*, 2014

Bioinformatics

- "Biochemistry and Biophysics Simulation Winter Course", By Dr. V. Guallar, Institute of Biochemistry and Biophysics (IBB), Tehran University, winter 2008
- "IPM-NUS" Workshop on Analysis and Applications of Protein Interaction Network", Institute for studies in theoretical Physics and Mathematics (IPM), Nov. 17-18 2008, Tehran. Iran

Business

- "Venture Challenge", Ecole Polytechnique Federale de Lausanne, Fall 2010
- "Information technology and e-business strategy", EPFL, 2013
- "Basics of Entrepreneurship", Entrepreneurship Center of Sharif U. of Tech, winter 2008
- "Business Planning", Entrepreneurship Center of Sharif U. of Tech, winter 2008
- "Principles of Negotiation", Entrepreneurship Center of Sharif U. of Tech, spring 2006

Academic & Professional Memberships

- Student Member of IEEE since January 2006
- Goalkeeper of electrical engineering futsal team, 2007
- Member of Sharif University astronomy group (2001 - 2003)
- Member of Sharif University photography group (2001 – 2003)
- Member of foundation board of Iran's Elites Association

Computer Skills

Programming Languages: Sufficient knowledge of MATLAB, C
Full knowledge of Assembly language of: 8051 family and AVR microcontrollers
Software and OS: NS-2 and OPNET, Linux (Basic knowledge), Microsoft Windows, CodevisionAVR, Microsoft Office (Word, PowerPoint, Visio, etc.)

Languages

Persian: Native (Reading, Writing, Speaking)

English: Fluent (Reading, Writing, Speaking), TOEFL (IBT): 114

French: Intermediate (Reading, Writing)

Arabic: Intermediate (Reading, Writing)

Chinese: Beginner, stopped progressing!

Hobbies

I am a **football** (soccer) fanatic (both playing and watching), love **driving** on deserted roads without a specific destination, do nature **photography** from time to time, love **astronomy** and enjoy **reading**, specially about philosophy and religion.