Clique-Based Neural Associative Memories

Asieh Abolpour Mofrad

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2021

UNIVERSITY OF BERGEN

Clique-Based Neural Associative Memories

Asieh Abolpour Mofrad

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 15.11.2021

Year: 2021

Title: Clique-Based Neural Associative Memories

Name: Asieh Abolpour Mofrad

Print: Skipnes Kommunikasjon / University of Bergen

Scientific Environment

This study is carried out at the department of Informatics, University of Bergen in the Selmer Center research group. I have been enrolled in the ICT Research School. I was under the main supervision of Prof. Matthew G. Parker, and co-supervision of Prof. Øyvind Ytrehus, and Prof. Anis Yazidi.

This thesis is dedicated to all the students who have been deprived of accomplishing their studies. Specifically, I commemorate the endearing memory of the Dasht-e-Barchi's schoolgirls who forever missed the opportunity of completing their school on May 8th, 2021.

Acknowledgements

Many wonderful people have supported me to accomplish this dissertation. The few paragraphs below are an attempt to express my deepest gratitude to all those who made this experience possible.

My sincere gratitude goes to my main supervisor Prof. Matthew G. Parker who made memorable my PhD journey at Bergen. I am grateful to Matthew for his valuable academic expertise and for introducing me to the self-dual graph-based codes and projective simulation agents. I am also particularly thankful to Matthew for supporting me to pursue my research interests. His positive attitude and sense of humor was always a source of inspiration for me. I would like to thank Prof. Øyvind Ytrehusfor for his mindful support during the final stages of my defense process. I am also truly honored to have Prof. Anis Yazidi as my co-supervisor. I thank him for his constant support and enthusiasm to accomplish this dissertation and his generosity to spend time finalizing the work.

I am also grateful to the members of the Selmer Center for a friendly and enjoyable environment. I especially extend my gratitude to Lilya Budaghyan (Group Leader) for being supportive. Besides, I thank all the management and administrative staff working at Department of Informatics particularly Pål Magnus Gunnestad, Michal Walicki, Linda Vagtskjold, and Tor M. Bastiansen.

Above all, I express my hearty gratitude to my parents, for all the support, love, and meaning they have gifted to me throughout my life. Without their support and encouragement, I would have never been at Bergen to pursue my studies. Likewise, I would like to appreciate my siblings Sajad, Samaneh, Khosrow, and my sister-in-law Ghazal for their unlimited kindness and support. I especially thank Samaneh for motivating me to accomplish this Ph.D. in addition to her contribution in Paper III of this thesis. This paper shows our joint efforts during July 2018, in addition to the 8-10 morning Zoom meetings within Fall 2020, during the Covid-19 pandemic.

Last but not least, my special thanks go to my husband Iman who has always been a caring companion. He has constantly encouraged me to stay steady and pursue my

objectives. Thanks for your considerate companionship through it all Iman!

Asieh Abolpour Mofrad
Oslo, May 2021

"I like crossing the imaginary boundaries people set up between different fields–it's very refreshing. There are lots of tools, and you don't know which one would work. It's about being optimistic and trying to connect things."

Maryam Mirzakhani, Winner of the Fields Medal.

Abstract

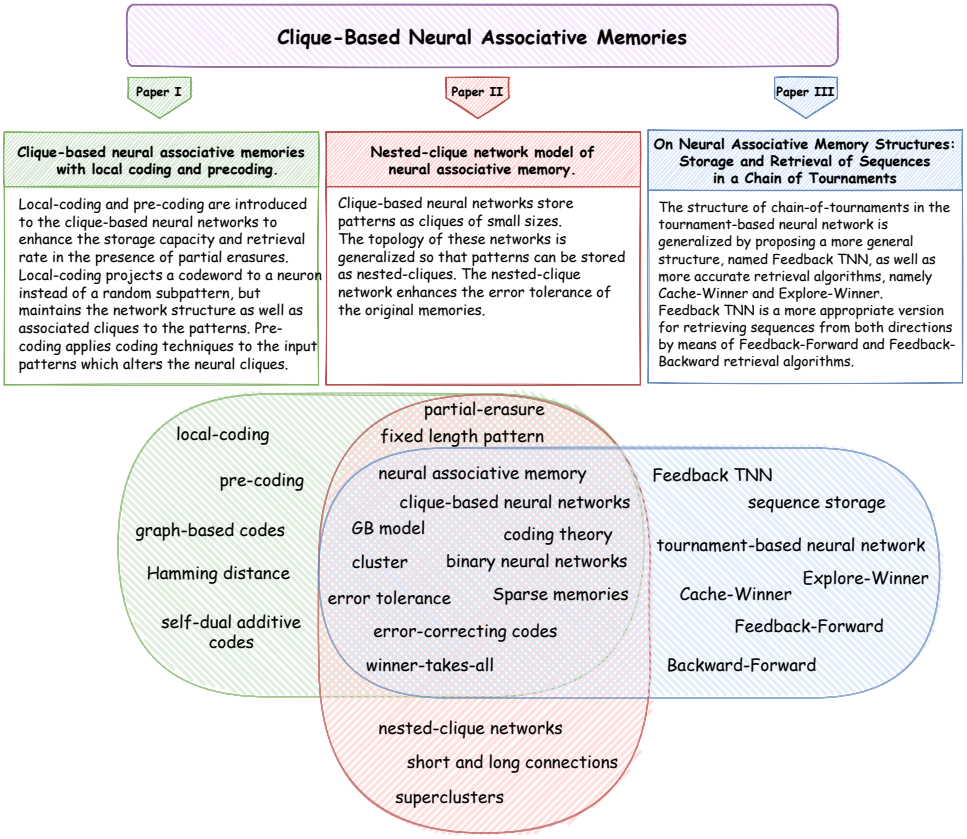
Auto-associative memories store a set of patterns and retrieve them by resorting to a part of their contents. This thesis focuses on developing and extending a type of associative memories relying on a sort of coded neural networks called clique-based neural networks.

Background. Both associative memories and erasure correcting decoders deal with similar tasks that revolve around retrieving missing pieces of information. However, despite the similarity of their respective tasks, there is a gap in terms of efficiency and performance which motivates applying coding techniques in the design of associative memories. Clique-based neural networks, introduced by Gripon and Berrou, denote a family of associative memories that are inspired by biological considerations as well as concepts from information theory. The usage of error-correcting coding and decoding techniques, borrowed from the field of information theory, considerably boosts the performance of these associative memories. The proposed neural network is organized in clusters of interacting neurons such that patterns can be stored as neural cliques, which in turn can be seen as codewords of a code. The tournament-based neural network is an extension of clique-based neural networks with the ability to store sequences. In this model, sequences of any length can be stored as chains of tournaments. Both clique-based and tournament-based associative memories have considerably larger storage capacity than the Hopfield model, which is commonly considered as the benchmark model for associative memories.

Contribution. The aim of this thesis is to advance the research area in associative memory by generalizing the concepts of clique-based and tournament-based neural networks. The generalization is expected to yield superior efficiency and retrieval performance. In this thesis, we use the following approaches. First, in Paper I, the coding techniques are used in two levels to enhance storage capacity and retrieval of partial erasures. In Paper II, a modification to the structure of clique-based neural networks is proposed to enhance the error-tolerance of the memory. Lastly, in Paper III, a modified version of tournament-based neural networks is used for retrieval of a sequence from a given segment by means of forward and backward retrievals. Moreover, the sequence retrieval performance is enhanced with the new retrieval techniques.

Discussion. We achieve the aim of generalizing the clique-based associative memories originally proposed by Gripon, Berrou, and co-authors to more resilient memories via using coding theory and graph theory approaches while maintaining their biologically plausible structures. The proposed models are quite flexible and can be employed collectively.

*K*eywords. Neural Associative Memory, Content Addressable Memory, Error Correcting Codes, Sparse Graphs, Sequence Storage, Clique-Based Neural Networks, Tournament-Based Neural Networks.



Thesis at a glance. As illustrated, the clique-based neural associative memories are generalized according to three directions that are published in Paper I, Paper II, and Paper III. A very brief summary of the contributions in each study as well as the central terms in this thesis are depicted.

List of Publications

This thesis consists of an introductory part and three scientific papers listed below:

1. **Mofrad, A. A.**, Parker, M. G., Ferdosi, Z., & Tadayon, M. H. (2016). *Clique-based neural associative memories with local coding and precoding.* Neural computation, **28(8)**, 1553-1573.*

2. **Mofrad, A. A.**, & Parker, M. G. (2017). *Nested-clique network model of neural associative memory.* Neural Computation, **29(6)**, 1681-1695.

3. **Mofrad, A. A.**, Mofrad, S. A., Yazidi, A., & Parker, M. G. (2021). *Neural Associative Memory Structures: Storage and Retrieval of Sequences in a Chain of Tournaments.* Accepted in Neural Computation.

*A preliminary version of this paper is presented in 2015 IEEE 14th Canadian Workshop on Information Theory (See Appendix A):

Mofrad, A. A., Ferdosi, Z., Parker, M. G., & Tadayon, M. H. (2015, July). *Neural network associative memories with local coding.* In 2015 IEEE 14th Canadian Workshop on Information Theory (CWIT) (pp. 178-181). IEEE.

Contents

Chapter 1

Introduction

1.1 Motivation

In the mid-twentieth century, the pioneering works of McCulloch and Pitts (1943) and Shannon (1948) opened the door in two seemingly distant fields respectively; artificial neural networks and information theory. Pursuing an information-theoretic perspective could bridge between the brain, or computational neuroscience, and information science. For instance, a variable-processor of a Low-Density Parity Check (LDPC) decoder [Gallager, 1962; MacKay and Neal, 1995] and a neuron in the McCulloch-Pitts model [McCulloch and Pitts, 1943] both aggregate input signals, positive/excitatory or negative/inhibitory, in order to generate output based on the summation result. These types of analogy can be an inspiring starting point in the design of neuro-inspired information-processing machines. The (artificial) neural associative memory, as a specific class of neural networks that mimic the associative memory in the brain, is capable of memorizing (learning) a set of patterns and recalling them afterwards in the presence of errors or erasures. This retrieval of missing pieces of information is similar to the problem of reliable communication over noisy channels, and more particularly to the iterative decoding techniques in modern coding theory and graph-based codes.

Despite the similarity of the tasks, there exists only a few studies that address the connections between information storage in the brain and modern error correcting decoders. Gripon and Berrou [2011] proposed a novel sparse neuro-inspired associative memory that regulates neurons into clusters and memorizes patterns as fully interconnected subgraphs or cliques in graph theory terms. This model of associative memory benefits from both information theory and neural network concepts where different approaches and strategies are used to combine error correcting codes with associative memories in

order to improve their performance [see, e.g., Gripon, 2011; Gripon and Berrou, 2012].

Learning and retrieval of temporal sequences in neural networks plays a significant role in human intelligence which is approached in many ways [see, Brea et al., 2011; Hawkins et al., 2009; Jiang et al., 2016; Maurer et al., 2005, to mention a few]. Tournament-based Neural Network [Jiang et al., 2016] is a high efficiency sequence model of the clique-based neural networks which benefits from oriented connections in the cliques.

Both the clique-based neural network [Gripon and Berrou, 2011] and the extended version for storing sequences [Jiang et al., 2016] can be considered as significant brain-inspired memory systems that initiate a wide range of studies in the associative memory research area.

In this thesis, the architecture of clique-based and tournament-based neural networks is improved and more accurate retrieval algorithms are proposed which yield a superior efficiency and retrieval performance. It is worthy to emphasize at the outset that the proposed models in this thesis are not intended to be faithful models of associative memory and neural networks but rather to use them as a source of inspiration. We try to achieve optimal designs of associative memory by applying techniques from coding theory and graph theory while their relevance to the memory system in brain is on an abstract level.

1.2 Organization

The rest of thesis is organized as follows. Chapter 2 gives a summary of required concepts from graph theory and neural associative memories. In Chapter 3, an overview of Coding theory and its applications to associative memories is given. Chapter 4 briefly explains the network of neural cliques and tournament-based neural networks as the ground of this thesis. A summary of the three papers is provided in Chapter 5. The thesis concludes by suggesting some future research directions in Chapter 6. The papers are included in Chapter 7, and finally a preliminary version of first paper which was presented in the 2015 IEEE 14th Canadian Workshop on Information Theory can be found in Appendix A.

Chapter 2

Graphs, Neural Networks, and Memories

2.1 Graph Theory

Graph theory as a branch of mathematics, on one hand investigates some of the deepest and most fundamental problems in pure mathematics, and on the other hand offers many useful results directly applicable to real world problems. Euler [1741] originally introduced graphs as an abstract way of capturing the fundamental properties of a specific problem domain, for instance, the relationships between the vertices, edges, and faces in geometric objects. In the rest of this section, all the necessary terms and definitions related to graph theory are provided in order to make the thesis self-contained [see, Bondy et al., 1976, for a through introduction to graph theory and its applications].

Formally, a graph G is an ordered pair (V, E) denoting an arbitrary nonempty set of objects V called vertices, and a set of edges $E \subset V \times V$, such that each edge is an unordered pair of, not necessarily distinct, vertices of G. The ends of an edge are *incident* with the edge, and vice versa. Two vertices that are incident with a common edge are *adjacent*. The same applies for two edges which are incident through a common vertex. An edge with distinct ends is called a *link*, while a *loop* refers to an edge with identical ends. The number of vertices adjacent to v is called the degree $d(v)$ of the vertex. A graph is *simple* if it has no loops and no multiple links between the same pair of vertices. Graph theory is mostly concerned with the study of simple graphs. A simple graph with n vertices that contains no edge is called an *empty graph*. It is called a *complete graph* and denoted by K_n if it has all the possible edges. Graph H is a *subgraph* of G (written $H \subseteq G$) if $V(H) \subseteq V(G)$, and $E(H) \subseteq E(G)$, where $E(H)$ is restricted to a subset of

edges that have both ends at $V(H)$. The *induced subgraph* of G on $W \subset V(G)$ (written $G[W]$) contains vertices W and all edges from $E(G)$ whose endpoints are both in W. A graph in which each edge (link) has an assigned orientation is called a *directed graph* and abbreviated as *digraph*. A directed edge (u, v) in a digraph that joins u to v is usually called *arc*, where u is the *tail* and v is the *head*. A *subdigraph*, can be defined similar to a subgraph. A clique is a complete subgraph of a simple graph, and a tournament is a directed graph captured by assigning an orientation to each edge in a complete (sub)graph. Graph G with n vertices can be specified by an $n \times n$ *adjacency matrix* $\mathbf{A}(G)[a_{ij}]$, where a_{ij} denotes the number of edges joining vertices i and j. In simple graphs, adjacency matrices are binary ($a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ otherwise) with all diagonal elements equal to 0. Moreover, the undirected simple graphs have symmetric adjacency matrices. The vertex set of a k-partite graph can be partitioned into k subsets in a way that no edge has both ends in the same partition. When $k = 2$, the graph is called *bipartite*. Figure 2.1 illustrates some of the above mentioned types of graphs. In the rest of thesis, we use the term graph and subgraph for directed graphs too whenever there is no ambiguity.

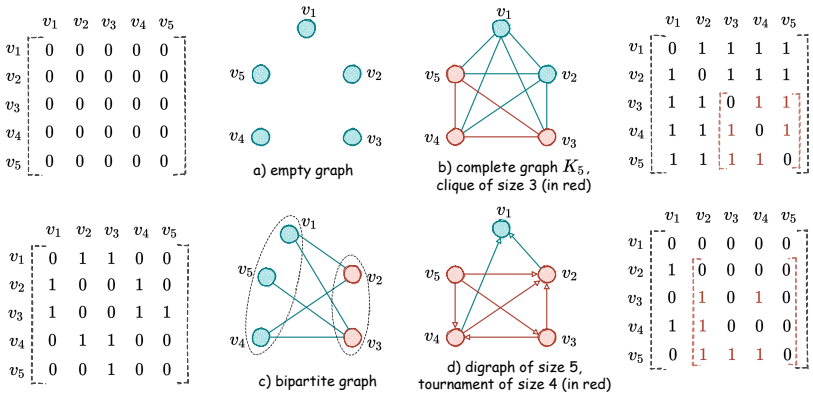


Figure 2.1: Illustration of an empty graph (a) with five vertices, Complete graph K_5 and a clique of size three (b), a bipartite graph (c), and a digraph with five vertices and a tournament of size four (d). The adjacency matrices of the graphs are also provided.

Graphs are mathematical objects and it is possible to define operations on two graphs to make a new graph. The *Cartesian product* of graphs G and H, denoted by $G \square H$, is a graph with vertex set

$$V(G \square H) = V(G) \times V(H),$$

namely, the set $\{(g, h)|g \in G, h \in H\}$. The edge set of the product is defined as all pairs $((g_1, h_1), (g_2, h_2))$ of vertices where either $(g_1, g_2) \in E(G)$ and $h_1 = h_2$, or $g_1 = g_2$ and

$(h_1, h_2) \in E(H)$[1] (see Figure 2.2 for an example).

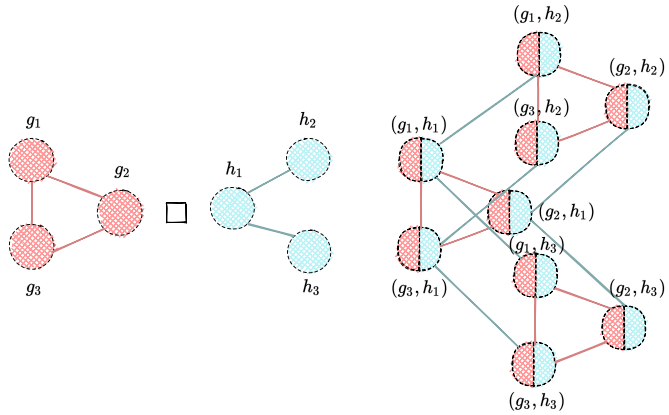


Figure 2.2: Illustration of Cartesian product of two graphs $G \square H$.

This Cartesian product is commutative up to isomorphism[2]. Remind that an operation is commutative if the order of operands does not affect the result, say $G \square H \approx H \square G$. This product is also connected with matrix operations and has many interesting applications in neural network structures [see, e.g., Imrich et al., 2008, for a depth study of Cartesian product in graphs].

Graphs can be used as a preliminary tool for representing many real-world situations such as a network of people or brain connectivity. Graphs play an essential role in design of a system, such as a communication scheme or an artificial neural network with specific characteristics. Graphs are used in the rest of thesis in the latter cases [see also, Sporns, 2018, for some applications of graph theory in brain networks].

2.2 Neural Networks

The field of Neural Networks studies the properties of networks of idealized *neurons*. One can study neural networks either to understand how the brain works (biology perspective), or to create machines that can *learn*, perform *pattern recognition* or *discover patterns in data* (engineering perspective), or just as an instance of complex adaptive systems with interesting properties. Neural networks, therefore, can be seen as computer implementation of interconnected nodes (processing units) and weights (connections) based loosely on the human brain. In a sense, they borrow ideas from brain and biological neural network functioning but usually are not faithful models of biological systems.

[1] We abuse the notation by representing edges, say (g_1, g_2), and vertices, say (g_1, h_1), similarly.
[2] A relabeling of vertices that keeps the graph structure

The first known artificial unit based on biological neurons is the *McCulloch-Pitts neuron* [McCulloch and Pitts, 1943] that functions as a logic gate. McCulloch-Pitts neuron gets active and sends a signal to other neurons in case it receives sufficient excitatory input which is not compensated by equally strong inhibitory input. *Perceptron* [Rosenblatt, 1958] is one of the first significant advances from the McCulloch-Pitts neuron that uses non-binary input and weights connections. The network can learn by adjusting the weights. A population of neurons perform a particular function in such a case. The learning via updating the weight is inspired by biological neural systems and a simple rule of synaptic plasticity, known as Hebb's rule [Hebb, 1949] or associative learning. Hebb's rule provides the basis for unsupervised learning [for experimental evidence of synaptic plasticity see, e.g., Karaminis and Thomas, 2012; Sommer, 2012]. The single neuron (perceptron), can be seen as a feedforward device in which the connections are pointing from the inputs to the output of the neuron. The neuron acts as a mathematical function with a weighted input sum that computes its output using a non-linear transform (see Figure 2.3).

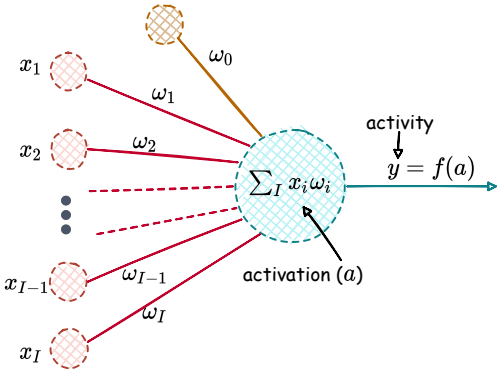


Figure 2.3: A single neuron (perceptron) architecture that has I inputs x_i and one output y. Each input is associated with a weight $\omega_i (i = 1, \cdots, I)$. ω_0 parameter refers to the *bias* that can be associated to an input x_0 which is fixed to 1. The *activation* of a neuron is $a = \sum_i \omega_i x_i$, and the output y which is called the *activity* of the neuron is a function of activation, i.e. $y = f(a)$ [see sections 39-41 of MacKay, 2003, for an in-depth study of single neurons].

Adding hidden layers to perceptrons yields multi-layer perceptrons, which includes input neurons, hidden neurons and output neurons. The multilayer perceptron is a feedforward network. All the connections in a feedforward network are directed so that the network forms a directed acyclic graph. Feedforward neural networks, like multilayer perceptrons, are common tools for nonlinear regression and classification problems [MacKay, 2003]. For multidimensional information processing such as memory, sequences and dynamics, feedback or recurrent neural networks are beneficial as they have cycles in their structures which can capture the relation and history of learning [see, e.g, Collobert et al., 2011;

Medsker and Jain, 1999, for some of applications of recurrent neural networks].

Several neural network models function as memories using simple learning algorithms [MacKay, 2003]. To devise a neural associative memory, the topology of the neural network, the learning process (updating weights between neurons), and the recalling algorithm have to be determined. The idea of associative memory will be discussed briefly in the next section followed by the Hopfield network as a fully connected recurrent network which is considered as the state-of-the-art reference of associative memories [see, e.g. Gripon and Berrou, 2011].

2.3 Associative Memories

The family of memories can be split into address-based (or indexed) memories and associative memories (or content-addressable) memories. Address-based memory, which is the more traditional one, stores data at a unique address and the data can be recalled through its complete unique address. In contrast, an associative memory compares a fragment of search data with stored data and returns the full matching data (see for e.g., section 38 of [MacKay, 2003]). *Association* denotes the connectivity of two or more pieces of information which is the basis of data retrieval. To be a little bit more specific, this model describes an auto-associative memory. In other words, auto-associative memory refers to all memories which are able to retrieve a piece of data from only a sample of itself. In hetero-associative memories the pair-wise relation between two patterns of different length, e.g. the name of an object and its picture, is memorized. Therefore Hetero-associative memories can recall an associated piece of datum from one category, when providing data from another category. Address-based memory is not robust and any mistake in the address can result into retrieval of a completely different memory. Moreover, it is not fault-tolerant and if an error happens on the data, say a bit is flipped, this error will be present whenever that memory is retrieved. The error-correction codes that detect and correct small numbers of errors make address-based memories error tolerance, however it can not be seen as an intrinsic property of the memory system [MacKay, 2003]. Biological memory systems are completely different from address-based memory systems which motivates the study of artificial neural networks as parallel distributed computational systems containing small processing units (neurons). Associative memories in computer science are indeed inspired from psychology and for this reason they are referred to with the same name in both fields.

2.3.1 Hopfield Neural Network

One of the first designs of an artificial neural network with auto-associative memory is the Hopfield neural network [Hopfield, 1982]. The Hopfield network is capable of pattern completion, given a partial pattern as well as error correction. The activity rule of the Hopfield network for each neuron is based on roughly updating its state similar to a single neuron with the threshold activation function.

$$x(a) = \Theta(a) \equiv \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0. \end{cases}$$

The Hopfield network is a recurrent network and every neuron's output is an input to all the other neurons (see Figure 2.4 for an illustration), therefore there is a need to specify an order for the network's updates. The updates could be *synchronous* or *asynchronous*. In the *synchronous* mode all neurons compute their activations and then update their states simultaneously. In the *asynchronous* update mode, a neuron at a time computes its activation and updates its state.

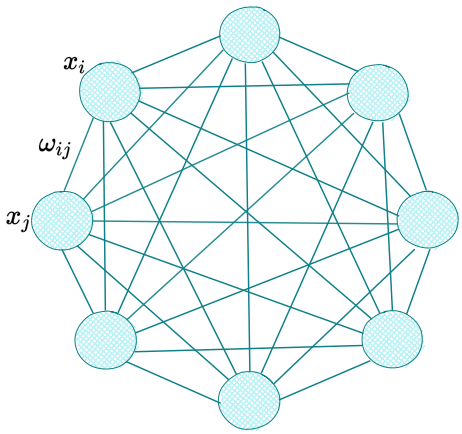


Figure 2.4: A Hopfield neural network consisting of eight neurons. The neurons are fully connected through symmetric, bidirectional connections with weights $\omega_{ij} = \omega_{ji}$. The activity, or the output of neuron i, is denoted by x_i.

The Hopfield network learning is based on Hebb's rule and the so-called *one-shot learning,* where the network requires to observe each memory only once to learn the association between them [Hopfield, 1982]. Each memory is a binary pattern, with $x_i \in \{-1, 1\}$. The learning rule is supposed to make a set of M binary patterns (or messages) $\mathbf{x}^m$ as

the stable states of the Hopfield network's activity rule, $\omega_{ii} = 0$ and

$$\omega_{ij} = \sum_{m=1}^{M} x_i^m x_j^m \text{ when } i \neq j.$$

The above rule is compatible with Hebb's rule since it increases the connection weight between two neurons if both have the same value. A distorted message (with values 0 for erased bits) can be retrieved in an iterative update of neuron's values (either synchronous or asynchronous) until the network reaches a fixed point; i.e. $x_i^{t+1} = x_i^t, i = i, \cdots, n$.

The fully connected structure of the Hopfield neural network is biologically implausible. Other drawbacks include low efficiency and spurious memories [see, Hoffmann, 2019, and references there]. Many modifications of the Hopfield neural network have been studied to overcome these downsides [see, e.g. Berrou and Gripon, 2010; Kim et al., 2017; Krotov and Hopfield, 2016; Maurer et al., 2005]. Boltzmann Machine, also named stochastic Hopfield network with hidden units, is a more powerful version that uses stochastic neurons [Ackley et al., 1985]. Willshaw-type model [Sommer and Palm, 1999; Willshaw et al., 1969] considers binary connections instead of weighted ones and sparse patterns. By virtue of *sparse coding* in the brain [see, e.g, Olshausen and Field, 2004; Rinkus, 2010, for sparse coding], sparse associative memories can be regarded as more biologically plausible models of memory [Gripon et al., 2016; Hoffmann, 2019]. Please note that sparse neural networks and neuro-inspired associative memories may refer to several categories [see, Gripon et al., 2016, for a comparative study of associative memories with sparse information].

Chapter 3

Error Correcting Codes and Associative Memories

3.1 Communication Over Noisy Channels

According to Claude Shannon, the father of Information Theory, "the fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point"[Shannon, 1948]. In order to use the channel efficiently, i.e. minimize transmission time and/or storage space, the source data is compressed using a source coding scheme. Considering the fact that communication channels are often noisy, error detection and correction codes add well-designed redundancy to the messages to overcome the channel noise, which is usually referred to as channel coding. Figure 3.1 illustrates a schematic diagram of a communication system and the source and channel coding principle.

Channel coding aims to make the noisy channel behave like a noiseless channel. An error correcting code is an encoding scheme that forms the allowed codewords (or patterns) for transmission through a noisy channel, in such a way that the received codewords which has been affected by noise can be still recovered. The receiver figures out whether the received word is an allowed one. If the received word is not an allowed one, the receiver finds the most probable codeword in the code. As a result, codes must be carefully constructed in order to have high efficiency in the sense that the largest possible distance between pairs of codewords is fulfilled.

Various strategies could be used to map the messages (or patterns) of length k into codewords of length n by adding redundancy. A key characteristic of a code is its

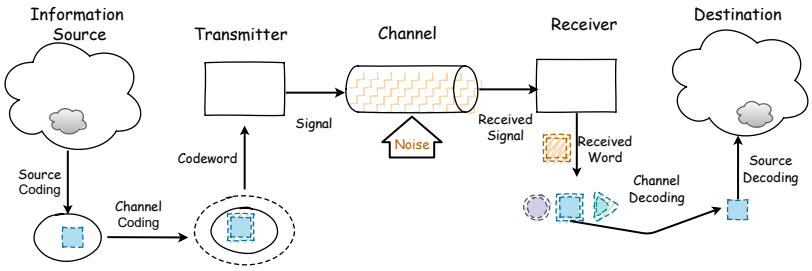


Figure 3.1: A schematic diagram of a communication system and the way source coding and channel coding provide efficient and reliable communication over noisy channel. The information source (or sender) starts the process by selecting a message to send. Source coding compresses the data, and channel coding adds redundancy to them. The transmitter (or encoder) converts the codeword into signals that can be sent from the sender to the receiver. The channel of communication is the infrastructure (or medium) that information passes through from transmitter to the receiver while noise might alter the transmitted information. The receiver performs the opposite process of transmitter. The Channel decoder finds the most probable emitted codeword and the source decoder reverses the process of compression to retrieve the initial message (or pattern).

minimum distance, that is defined as the lowest Hamming distance between any two codewords in the code. The Hamming distance between two codewords is defined as the number of positions with different components. The minimum distance is closely linked with the maximum number of errors that is guaranteed to be corrected. More precisely, the maximum number of errors that is guaranteed to be corrected in any codeword belonging to a code with minimum distance d_{min} equals $\lfloor \frac{d_{min}-1}{2} \rfloor$. In the case of erasures, this value equals $d_{min} - 1$. The decoder chooses the closest codeword to the received word in terms of Hamming distance as it is supposed to be the most likely transmitted codeword.

3.2 Role of Coding theory in Associative memories

Shannon's well-known "source coding - channel coding" scheme of communication has similarities to how the brain captures and stores essential information (see Figure 3.2).

The source coding part, that is concerned with the acquisition and compression of data, has been vastly studied by computational neuroscience community under the name of *machine learning* which mimics the brain as an intelligent learner and classifier [see Berrou et al., 2014, and references therein]. The channel coding part, which is analogous to the way mental information is encoded, retrieved and propagated in a robust and durable manner, is the focus of this thesis.

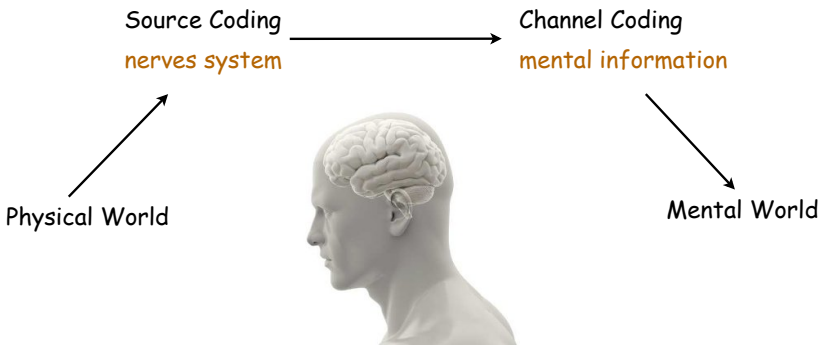


Figure 3.2: A schematic diagram of Shannon's model of cognition [taken from Berrou et al., 2014, Fig.1]. The external world provides richly detailed information. The information is strongly compressed first and then some *smart redundancy* is added which enables robust and durable memorization.

In a neural associative memory, a set of patterns are learnt (memorized) first, and then the full pattern that matches a given noisy fragment of a learned pattern is retrieved. In channel coding, on the other hand, reliable and efficient retrieval of a set of codewords (analogous to patterns in associative memory terminology) from a noisy version that the receiver receives from the channel is expected. Coding theory techniques can increase the storage capacity and improve the error correction capability of associative memories. Neural associative memories are usually able to memorize any set of randomly chosen patterns without distance optimization. Coding theory, therefore, can improve neural associative memories error tolerance [see, Berrou and Gripon, 2010; Berrou et al., 2014; Gripon, 2011; Hopfield, 2008; Mofrad et al., 2016; Salavati, 2014, to mention a few].

One approach that associative memory can benefit from coding theory is to focus on learning patterns that have some kind of inherent redundancy. Berrou and Gripon [2010] achieved considerable improvements in the pattern retrieval capacity of Hopfield networks, by using error correcting codes combined with sparse data representation. Salavati et al. [2011] proposed a neural association mechanism that utilizes binary neurons to memorize patterns belonging to Gold sequences, which constitute a family of low correlation sequences. These techniques, that involve some kind of pre-coding, all enhance pattern retrieval capacity of the associative memory. It is noteworthy that working with structured patterns is more biologically meaningful due to the fact that sensory inputs to the brain are pre-processed before actually being stored [see, e.g., Berrou et al., 2014; Salavati, 2014; Salavati et al., 2011].

In another line of approaches, the associative memory is designed to be able to memorize any random set of patterns. To benefit from coding techniques and redundancy, the memory structure can impose some rules that make the memory more error resistant.

For instance, as suggested in [Gripon and Berrou, 2011; Hopfield, 2008], the neurons can be organized in clusters and just one neuron per cluster could be present in memorizing a pattern. As another example, Salavati and Karbasi [2012] designed a two-level neural associative model inspired from graph-based codes, in which the pattern neurons are partitioned into clusters of bipartite graphs, where sub-patterns should form a subspace or a code in coding terminology. These approaches can be seen as a sort of coding in the memory structure design or local coding.

Chapter 4

Networks of Neural Cliques

By virtue of *sparse coding* in the brain [see, e.g, Olshausen and Field, 2004; Rinkus, 2010, for sparse coding], sparse associative memories can be regarded as more biologically plausible models of memory [Gripon et al., 2016; Hoffmann, 2019]. Sparse neural networks and neuro-inspired associative memories may refer to several categories. Gripon et al. [2016] have studied associative memories with sparse information, where patterns in the learning set are random strings of 0s and 1s with about $\log n$ 1s, only. The sparse clique-based associative memory models which are the ground models of this thesis are covered in the subsequent sections.

4.1 The Original Clique-Based Neural Networks

The Clique-Based Neural Networks (CBNN) introduced by Gripon and Berrou [2011], also referred to as GB model, are based on Willshaw networks [Willshaw et al., 1969], and constructed by dividing a neural network with n neurons into c clusters which might have different sizes [see also, Hopfield, 2008, for another clique-based neural associative memory]. The patterns to be memorized are chosen in such a way that only one neuron in each cluster is active for a given pattern[1]. A pattern then can be considered as a random vector of length $c \log(n/c)$. The storage process maps a pattern to its associated neurons, activates them and then connects all of them to form a clique (a complete subgraph). The input patterns are formed from a pre-defined alphabet $\mathcal{A}$ and each neuron represents one of the symbols in the alphabet. For the sake of simplicity, the size of all clusters are considered identical and equal to $l = n/c$, and therefore the alphabet size

[1]The term 'fanal' (which means lantern or beacon) is used by the authors [Gripon and Berrou, 2011] to highlight the uniqueness of an active neuron in each cluster.

is equal to $l = |\mathcal{A}| = n/c$. Furthermore, in order to ease working with binary patterns, we let $l = 2^{\kappa}$ and each binary pattern of length κ is then assigned to a symbol in the alphabet $\mathcal{A}$ or equivalently a unique neuron in a cluster. The function $f(.)$ maps each subpattern to a unique neuron in the corresponding cluster,

$$f : \{0,1\}^{\kappa} \rightarrow [|1; l|].$$

where $[|1; l|]$ refers to the integer numbers between 1 and l, inclusive. The j^{th} neuron in the i^{th} cluster is denoted by n_{ij} where its associated value, $v(n_{ij})$, equals one when it is active, and zero otherwise; $1 \leq i \leq c$ and $1 \leq j \leq l$.

4.1.1 Learning or Storage Process

The set of patterns to be stored, or learned by the network, is denoted by $\mathcal{P}$ where a pattern $p \in \mathcal{P}$ contains c sub-patterns, $p = p_1 p_2 \cdots p_c$; for $p_i \in \mathcal{A}$, $1 \leq i \leq c$. The learning process entails the assignment of a set of unique neurons -one per cluster- to each pattern $p \in \mathcal{P}$:

$$p = p_1 p_2 \cdots p_c \rightarrow (f(p_1), f(p_2), \cdots, f(p_c))$$

$$\text{where} \quad f : \{p_i\} \rightarrow \{n_{ij} | 1 \leq j \leq l\}.$$

The selected neurons get activated, $v(n_{ij}) = 1$, and a clique is formed by connecting these c active neurons to each other through binary edges (i.e. with weight 0 or 1). Accordingly, the learning process generates a set of binary edges

$$\mathcal{W} = \{\omega_{(ij)(i'j')} | \text{ if } i \neq i' \text{ and } \exists\, p \in \mathcal{P} \text{ s.t. } f(p_i) = n_{ij} \text{ and } f(p_{i'}) = n_{i'j'}\},$$

where $\omega_{(ij)(i'j')}$ is an edge between n_{ij} and $n_{i'j'}$. It is noteworthy that an edge $\omega_{(ij)(i'j')}$ belongs to $\mathcal{W}$ if at least one pattern has both n_{ij} and $n_{i'j'}$ neurons; however, beyond one, the number of patterns that meet this criterion does not affect the set $\mathcal{W}$.

Figure 4.1 illustrates the storing process in a GB network with 64 neurons split into four clusters of 16 neurons each. The binary pattern $p = 0100000110001100$ is split into 4 sub-patterns, namely, $p_1 = 0100$, $p_2 = 0001$, $p_3 = 1000$, and $p_4 = 1100$. Then, each of these sub-patterns is mapped to a unique neuron in the corresponding cluster; $f(p_1) = n_{1,5}$, $f(p_2) = n_{2,2}$, $f(p_3) = n_{3,9}$, and $f(p_4) = n_{4,13}$. Finally, these neurons are fully interconnected to build a clique and store the pattern in the network (the yellow clique in Figure 4.1 represents this pattern).

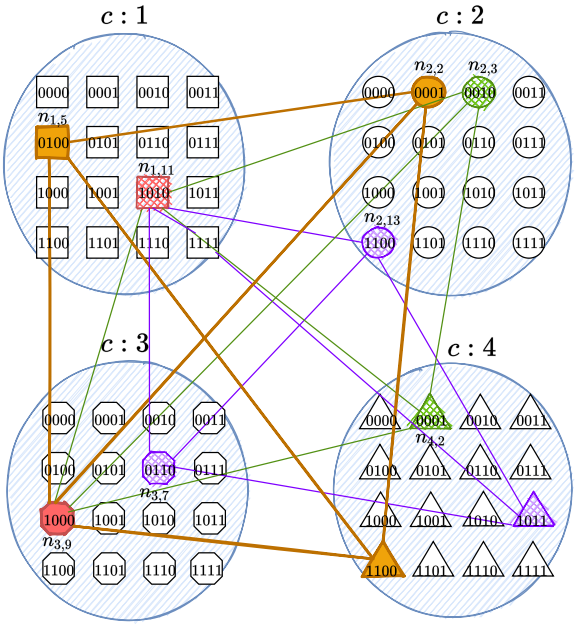


Figure 4.1: Learning of three patterns, in a CBNN structure with $c = 4$ clusters and $l = 16$ neurons per cluster. Node $n_{i,j}$ refers to the j^{th} neuron in the i^{th} cluster. The three patterns $(4, 1, 8, 12)$, $(10, 2, 8, 1)$, and $(10, 12, 6, 11)$ maps into cliques with yellow, green, and purple colors respectively. The red nodes, $n_{1,11}$ and $n_{3,9}$, belong to two patterns.

4.1.2 Retrieval Process

Given a partial pattern, the purpose of the retrieval is to obtain a complete pattern. The retrieval process utilizes the stored connections during the learning phase to restore the erased or erroneous data related to a stored pattern. Different retrieval methods might be used according to the type of distortion [see, Aboudib et al., 2014, for instance]. The retrieval procedure from an erased pattern consists of two stages: the *global dynamics* stage which establishes or eliminates connections based on provided parts of the pattern, and a *local decision* stage that will decide about activation of neurons following a rule. The Winner-Takes-All rule activates neurons with the highest activity (or maximum score) whilst Losers-Kicked-Out rule (LsKO) uses a threshold filter to eliminate active neurons with less activity [see Jiang, 2014, for an overview of different retrieval algorithms and activation rules]. This global and local retrieval gradually complete the clique and as a consequence retrieve the pattern.

Clearly, higher density (which is defined as the the ratio between the number of established connections within the storage process and all possible connections) negatively affects the retrieval accuracy. In Figure 4.1, given each of the coloured nodes with yel-

low, green and purple, a unique retrieval is possible. In contrast, it is impossible to retrieve a pattern by finding a unique clique using only one of the red nodes.

Various extensions of the CBNN family of associative memory have been proposed in the literature, leading to improvements in data storage and retrieval. For instance, Aliabadi et al. [2014] proposed an even more sparse structure that maps a sparse pattern to a subset of clusters. Retrieval algorithms for sparse patterns in networks of neural cliques are studied by Aboudib et al. [2014]. The latter work was improved to deal with the challenging scenario of high interference leading to significantly corrupted probe by Jiang et al. [2015]. We refer the interested reader to the following references for extensions and applications of clique-based neural networks [Aboudib et al., 2016; Berrou and Kim-Dufor, 2018; Danilo et al., 2015; Hacene et al., 2017, 2019; Jarollahi et al., 2014, 2015; Jiang et al., 2016; Larras and Frappé, 2020; Larras et al., 2018; Marques et al., 2017; Yao et al., 2014]

4.2 Tournament-based Neural Network for Sequence Storage

A Tournament-Based neural network (TNN) is an extension of the non oriented clique-based neural network which is able to store sequences of arbitrary length in binary neural networks [Jiang, 2014; Jiang et al., 2016]. In a chain of tournaments of order c and degree r, denoted by $\mathcal{T}_r(c)$, each of c nodes is connected to the next r nodes through directed edges and therefore make a tournament of size $r+1$. A TNN then can be understood as a concatenation of tournaments of size $r+1$ consisting of consecutive neighbors (see Figure 4.2, for an illustration).

In the following, the storage process of a sequence $s \in \mathcal{S}$ with L component is explained, where $s = s_1 s_2 \cdots s_L$; for $s_t \in \mathcal{A},\ t = 1, 2, \ldots, L$, and $|\mathcal{A}| = l$ and $\mathcal{S}$ is the set of all sequences to be stored. We suppose that the clusters are labeled from 1 to c. A unique sequence of neurons must be assigned to the sequence using function $f = (f_1, \cdots, f_c)$, where $f_i,\ i = (t-1 \bmod c) + 1$, maps a component s_t, to the neuron n_{ij} in cluster i:

$$f_i : \{s_t\} \rightarrow \{n_{ij} | 1 \leq j \leq l, \},\ 1 \leq i \leq c,$$

accordingly,

$$f(s) = (f_1(s_1), f_2(s_2), \cdots, f_c(s_c), \cdots, f_{(L-1 \bmod c)+1}(s_L)).$$

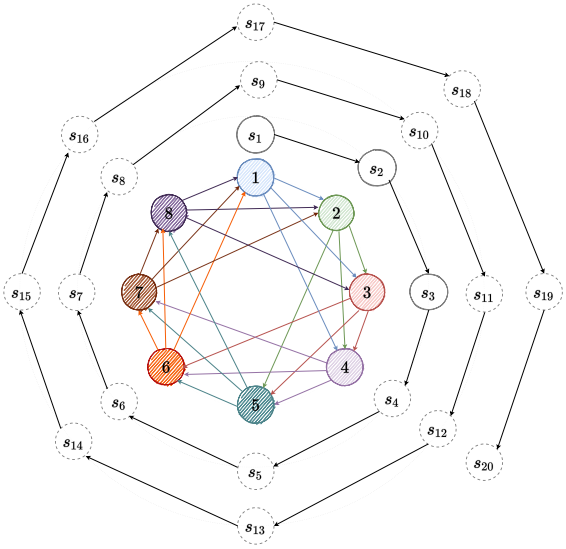


Figure 4.2: A chain of eight tournaments, $\mathcal{T}_3(8)$, of size $r+1=4$, storing sequences of length 20. The colored nodes represent clusters, and directed edges represent available connections between nodes inside the clusters. This figure is taken from paper III which is based on [Jiang et al., 2016, Fig. 5].

Learning proceeds by connecting neuron n_{ij} to neuron $n_{i'j'}$ at passage π in the following manner:

$$n_{ij} \to n_{i'j'}, \quad \text{if:} \begin{cases} f_i(s_{(i+(\pi-1)c)}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{and, } 1 \leq \delta_i(i') \leq r$$

where $\delta_i(i') = (i'-i) \bmod c$, and $1 \leq \pi \leq \lfloor \frac{L}{c} \rfloor$.

Altogether, for a given $s \in \mathcal{S}$, if the previous conditions are satisfied for passage π such that $n_{ij} \to n_{i'j'}$, we define $N_{s,\pi}(n_{ij}, n_{i'j'}) = 1$, as a result of which n_{ij} is connected to $n_{i'j'}$, in sequence s, alternatively, we define $N_{s,\pi}(n_{ij}, n_{i'j'}) = 0$. For instance, for depicted sequence s in Figure 4.2, the associated neuron to s_3 is connected to the associated neurons to s_4, s_5 and s_6 in passage $\pi = 1$, but not to the associated neurons to s_{12}, s_{13}, and s_{14}, (in passage $\pi = 2$), or s_{20} (in passage $\pi = 3$). So the neighboring connections are defined in accordance with both s and π values. The network possesses the following connections by the end of learning process,

$$\mathcal{W} = \{\omega_{(ij)(i'j')} | \text{ if } \exists\ s \in \mathcal{S}, \text{ and } \exists\ \pi \in [1 : \lfloor \frac{L}{c} \rfloor] \text{ s.t. } N_{s,\pi}(n_{ij}, n_{i'j'}) = 1\}$$

where $\omega_{(ij)(i'j')}$ is a directed edge from n_{ij} to $n_{i'j'}$ and $1 \leq i, i' \leq c,\ 1 \leq j, j' \leq l$.

The retrieval process may commence with any subsequence of r consecutive components of a previously stored sequence s. If the given subsequence is not from the beginning of the sequence, the extra information of the associated clusters is required. The first three components of the sequence s, i.e. s_1, s_2, and s_3, are shown with solid circles in Figure 4.2 to represent the given part of sequence, and the rest of components that should be retrieved by the retrieval process with dashed circles. The retrieval procedure is sequential employing a Winner-Takes-All decision at each cluster where the activation of a component relies on the connections from r previous clusters.

Chapter 5

Summary of Papers

The studies included in this thesis propose neural auto-associative models for the storage and retrieval of messages and sequences. In the following, a summary of the papers is provided.

5.1 Paper I

The first paper, entitled "Clique-based neural associative memories with local coding and precoding" is published in Neural computation (2016), MIT press.

This paper focuses on improving storage capacity and retrieval performance of clique-based neural associative memories (introduced by Gripon and Berrou [2011]) in the presence of partial erasures by applying local coding and pre-coding techniques. The increase in the retrieval capacity in both techniques is verified through simulations.

The local coding proposed in this study keeps the number of neurons in the associative memory fixed, but turns each sub-pattern to a codeword by adding some redundancy before learning the pattern. The codewords are then mapped into the neurons in clusters (see part b) in Figure 5.1). By using the local-coding, the edge set W remains unchanged, but in the retrieval, the distance between codewords within clusters enhances the performance at the local level. Different codes with possibly distinct alphabets and codebook sizes for each cluster can be considered in local coding. In the pre-coding technique proposed in this paper, first, a general coding (or a pre-coding) is applied to each pattern. Then, based on the CBNN learning process, the codeword is split into sub-codewords that are mapped to a neuron in the corresponding cluster and memorized (see Figure 5.1 part c)). Pre-coding prevents two near patterns from being

members of the learning set by maximizing the minimum pairwise distance in the learning set which enhances error-tolerance of the memory. Since patterns are transformed to codewords, pre-coding requires changes in the network structure, and therefore it does not preserve the associated cliques with the learning set in the uncoded network. In summary, local coding limits the possible matching set in the clusters, and pre-coding forces patterns to be well separated and consequently increases distance in cliques. It is noteworthy that we can consider a general framework that includes all the three scenarios depicted in Figure 5.1. In this way, a pattern is mapped into a codeword via a pre-coding first. The codeword is split into appropriate sub-patterns where a local coding is used to increase the distance between the neurons. Finally, the associated clique is formed by connecting the active neurons. Pre-coding and local coding could be unitary code (no function on the input) or be an arbitrary code.

A preliminary version of local coding method was presented at CWIT [Mofrad et al., 2015] where a more efficient retrieval algorithm is proposed that improves the memory resistance in partial erasures. As illustrated in Figure 5.2, the algorithm benefits from local coding in the local decision stage (or local check) since it is based on the erasure tolerance of the local coding technique (and to a certain extent pre-coding). The pre-coding that alters the connections and cliques' distances, is beneficial in the iterative part of the algorithm (or the global dynamics stage) and the last phase of searching for cliques. Therefore, a more reliable memory is achievable by combining both techniques together.

To demonstrate that the local coding and pre-coding techniques can benefit from various coding techniques, two error correcting codes are used in the simulation. The algebraic Reed-Solomon (RS) code [Reed and Solomon, 1960] is resorted to as the first option. RS is widely used for data storage and is able to detect and correct combinations of errors and erasures. We also used as a second option self-dual additive codes over field $GF(4)$ that have many interesting properties such as simple-graph representation [see, Danielsen, 2008, for instance]. The simple graph representation of these codes can be further used in the design of neuro-inspired memories with features like retrieval with message-passing on the simple graphs. It is noteworthy that, unlike self-dual additive codes over $GF(4)$, usually graph-based codes with bipartite representations are used in neuro-inspired memories [see, Berrou and Gripon, 2010; Salavati, 2014, for instance]. Retrieval results in presence of partial erasures yield dramatically better error-tolerance using either local coding or pre-coding techniques which suggests a combined version in the case where more data protection is needed. Moreover, the results suggest that the local coding can better handle higher erasure probabilities while pre-coding is more appropriate for larger sizes of the learning set.

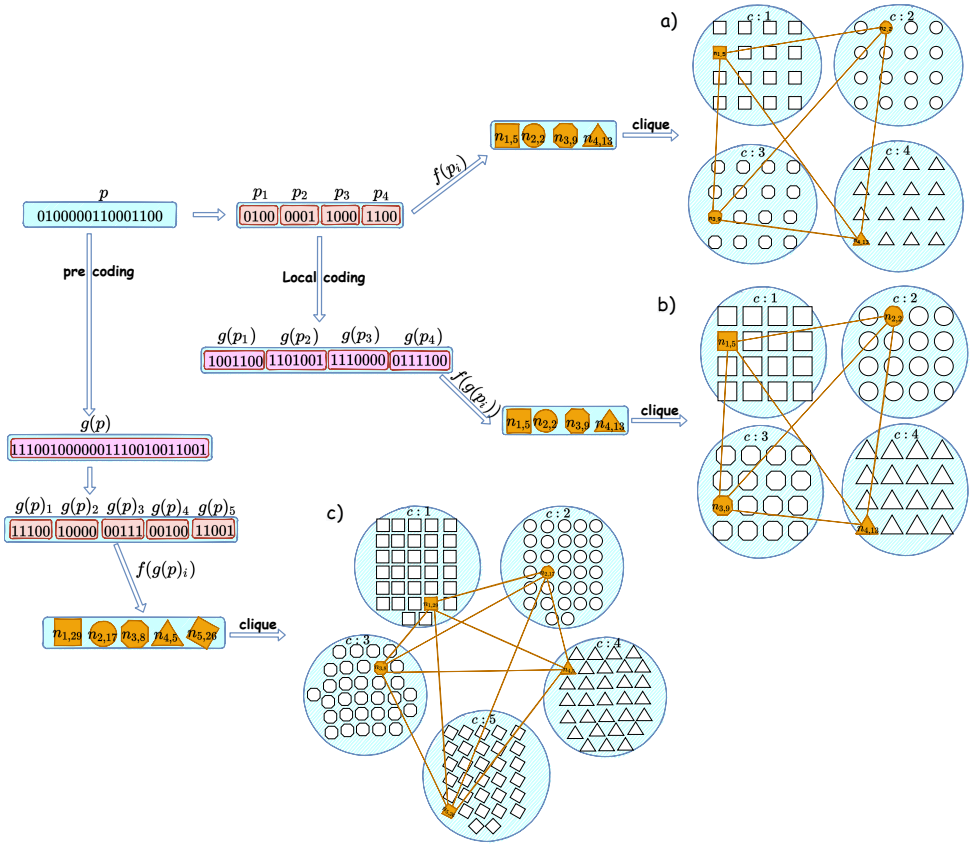


Figure 5.1: An illustration of learning binary pattern $p = 0100000110001100$, in a CBNN structure with three scenarios; a) the original CBNN, b) local coding, and c) pre-coding. In a) the pattern is split into 4 sub-patterns, which are mapped into the neurons in the $c = 4$ clusters. The learning is to connect these nodes and make a clique. In b) each sub-pattern p_i is encoded to a codeword $g(p_i)$ and then mapped into the associated neuron. The structure of the network and the associated clique do not change in the local coding, but the address of each neuron is changed since a codeword is mapped into a neuron instead of a raw sub-pattern (shown with larger node sizes). The pre-coding converts the pattern p to a codeword $g(p)$ first, and then splits it into sub-patterns and maps it into a clique. As shown in c) the pre-coding changes the structure of the network and the clique which is depicted with more clusters and more neurons per cluster.

5.2 Paper II

The second paper, entitled "Nested-Clique Network Model of Neural Associative Memory", published in Neural Computation (2017), MIT press.

Paper II generalizes the CBNN architecture by associating a nested-clique to each pattern instead of a clique. The nested-clique structure can be viewed as a two-layer network

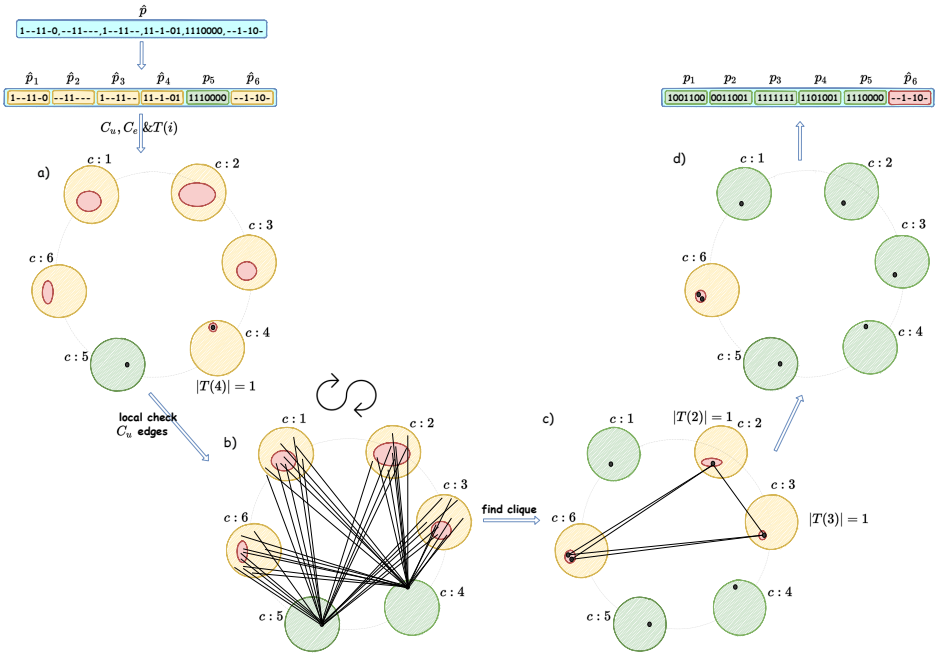


Figure 5.2: An illustration of the retrieval algorithm of a partially erased pattern $\hat{p}$. In a) the algorithm first splits the clusters into unerrored C_u and errored C_e sets and initializes $T(i)$ sets in the errored clusters that contain possible candidates (or winners). The local check is searching for $T(i)$ sets with one member and decodes the erasure at the cluster level. In b) the algorithm starts iterative retrieval by removing candidates that are not connected to all active neurons in C_u clusters (green clusters). The algorithm stops iterations if all the $T(i)$ sets have more than one component and the edges from C_u can not further reduce the size of candidates. In this case, the algorithm searches for a clique in the candidates in $T(i)$ sets for $i \in C_e$. In c) a case with two alternative cliques is presented and therefore no unique retrieval is found. The algorithm updates the components when $|T(i)| = 1$ (cluster 2 and 3 in part d)) and then maps the active neurons to the closest match.

where two clique-based auto-association networks are intertwined. The nested-clique structure enhances the error tolerance of the network and provides the network with one more controlling parameter; the configuration of clusters within superclusters.

A nested-clique neural network splits the neurons into $c = c_1 \times c_2$ clusters and groups the clusters into c_1 superclusters of size c_2. The original clique-based model can be considered as a special case when $c_1 = 1$ and $c_2 = c$. To store a pattern, the first step of mapping a pattern to the associated neurons is similar to the clique-based structure. The channels of allowed connections between clusters can be defined by a Cartesian product between a c_2-clique of clusters and a c_1-clique of superclusters (see part a) in Figure 5.3). Therefore, to make a nested-clique, the learning proceeds by establishing connections between active

neurons in each of superclusters (see yellow *short connections* in part b) of Figure 5.3), and also forming c_2 cliques among equivalent clusters in different superclusters (see *long connections* between the three superclusters in part b) of Figure 5.3). The Cartesian product can explain some of the properties of the nested-clique neural networks such as the unique topology of c_1-cliques-of-c_2-cliques and c_2-cliques-of-c_2-cliques (commutative operation).

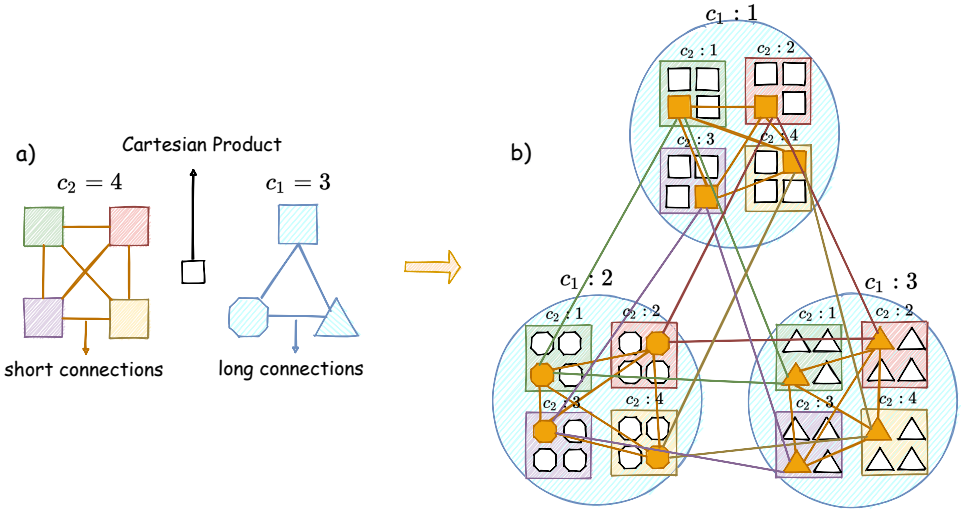


Figure 5.3: a) represents the configuration of clusters using a Cartesian product operation between a clique of 4 clusters and a clique of 3 superclusters. The short connections and long connections indicate which active neurons must be connected. In b) the structure of the network and a learnt pattern in the form of nested-cliques in a network with 48 neurons, split into 3 superclusters each with 4 clusters of 4 neurons, are depicted. In each supercluster a clique of size 4 is established by short connections; also activated neurons in the equivalent clusters (same colours) are connected with long connections to form cliques of size 3. Therefore, the patterns are stored as 3-cliques-of-4-cliques or $K_3[K_4]$.

Retrieval is basically similar to the clique-based version. The difference is at the second stage where cliques are constructed to kick out neurons from candidate sets in which the algorithm must consider both short and long connections. Using both short and long connections in the retrieval process is beneficial and conveys the information between clusters that are not connected directly.

When it comes to comparing the CBNN model and the nested-clique, it is noteworthy that a nested-clique network can be obtained from a CBNN (or a set of CBNN) network(s) by adding extra connections. In this case, the nested-clique network would be stronger and more robust against erasure. From the other side, a nested-clique network can be realised from a CBNN network by removing some connections and degenerating

a clique-based network. The nested-clique network is less robust to erasure in such a case. Therefore, in order to achieve a a fair comparison, the nested-clique structure and the original clique-based model are simulated under similar amounts of available memory or equivalently the number of possible connections. The nested-clique network shows better error-tolerance in both cases of fixing capacity or erasure rate, but its diversity, i.e. the number of learnt patterns, is lower than the original network. The reason is that nested-clique structure requires more clusters and neurons to use equal number of connections as clique-based and therefore the nested-clique model learns longer patterns whilst its diversity (due to fixed capacity) is smaller. In consequence, for a fixed amount of memory used, if the concern is the total information that network is able to memorize and then retrieve in presence of partial-erasure, nested-clique is superior. On the other hand, since clique-based network memorizes smaller patterns, it can learn and recall more patterns than the nested-clique structure and must be chosen when diversity is the concern.

Several comparisons between different nested-clique scenarios for a fixed number of neurons are also reported. The results suggest avoiding very small sizes of clusters in general, and determining the number of superclusters based on other conditions, say fixed erasure rate, or fixed capacity.

5.3 Paper III

The third paper, entitled "On Neural Associative Memory Structures: Storage and Retrieval of Sequences in a Chain of Tournaments", accepted to be published in Neural Computation, MIT press.

Paper III addresses the storage and retrieval of sequences in sparse binary neural networks. Feedback Tournament-Based Neural Network (Feedback TNN) is the proposed storage architecture that is a more general form of TNN [Jiang et al., 2016] that was addressed in section 4.2. The required algorithms for storage and retrieval in Feedback TNN together with two higher performance retrieval algorithms, Cache-Winner and Explore-Winner are proposed. The Feedback TNN structure and new retrieval algorithms contribute to the field of aut-associative memories and offer appreciable gains, highlighted by simulations.

In Feedback TNN structure, the components of a sequence are associated forward and backward respectively to the following and previous components in the sequence (see part a) of Figure 5.5). This re-configuration of connections in the chain of tournaments assists

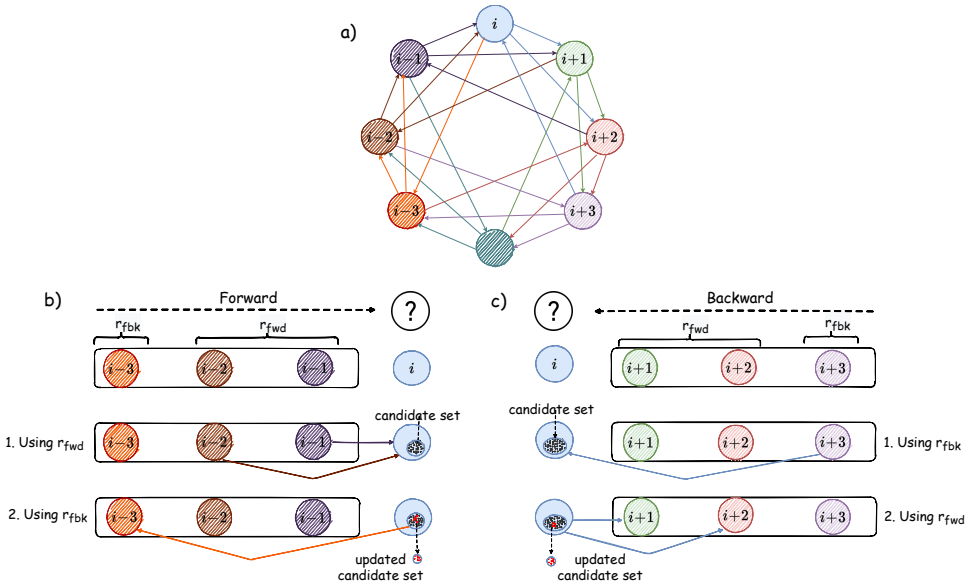


Figure 5.4: A chain of tournament structure with feedback connections is illustrated in part a) where the first two connections of each tournament are clockwise and the last connection is counterclockwise. Forward and Backward retrieval processes are illustrated respectively in part b) and c) when a segment of 3 components are given. This figure is a modified version of Figures 5 and 6 in Paper II.

backward retrieval, and therefore any large-enough segment of a sequence is sufficient to retrieve the whole sequence. In this case, a pre-matching process is necessary to locate the clusters on which the given sequence segment was originally stored. Feedback-Forward and Feedback-Backward retrieval algorithms are addressing the retrieval in Feedback TNN, which are compatible with original TNN, as a particular case. As depicted in part b) of Figure 5.5 for Feedback-Forward, the first step that uses r_{fwd} is relevant for TNN. The Backward algorithm initiates a candidate set of size l for each component and builds a sub-graph with the next $r_{fwd} = r$ components; in part c) of Figure 5.5, simply the second step which uses r_{fwd} is applicable for TNN. Therefore, despite that the Feedback-Backward retrieval is proposed for Feedback TNN structure, it is well-functioning with the original TNN, as a special case with zero feedback connections.

Cache-Winner and Explore-Winner are the two retrieval algorithms introduced to increase the retrieval performance. In Cache-Winner, a large part of the ambiguities that can occur at retrieval of a component is removed by taking up possible and abandoned paths from the results of previous steps, temporarily stored in a cache memory. In case an error is detected during retrieval, the Cache-Winner revisits the cache memory and changes some previous randomly selected components. Explore-Winner, on the other hand, anticipates what might happen in future steps before making a decision at the

current step of retrieval. In this way, the randomness in decisions is reduced by considering the consequences of each decision. Therefore, while Cache-Winner algorithm tries to resolve the mistake decisions when realizing them, Explore-Winner tries to avoid mistakes as early as possible.

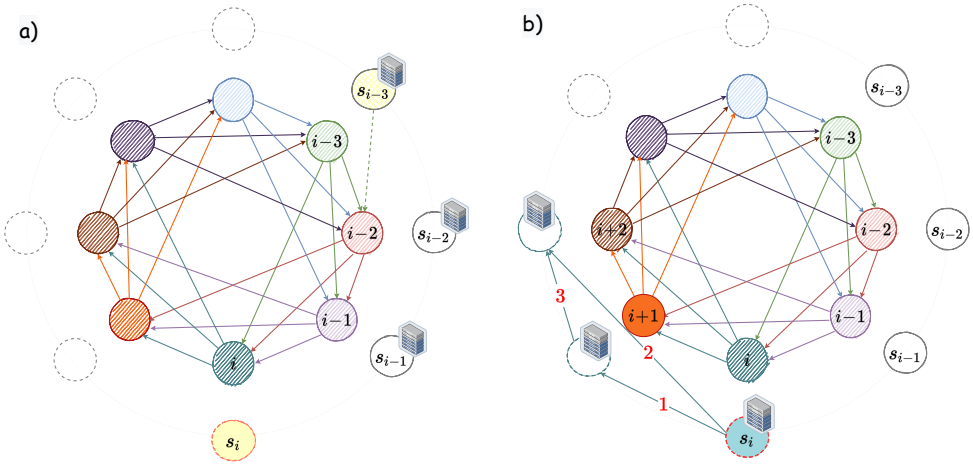


Figure 5.5: The mechanisms of using temporary cache memory and exploration technique are illustrated in parts a) and b) respectively. In a) the component s_i represents the situation where no candidate with the full score is found. This detects an error in the retrieval of the previous $r = 3$ components. The cache memory is used to revise the previous random decisions and try another neuron to be activated. In b) the component s_i represents the situation where more than one full score node (winner) is found for activation in cluster i. The exploration technique tries to eliminate the number of randomly chosen components among the winners by investigating the results of each candidate selection, and by eliminating inappropriate choices. This figure is a modified version of Figures 3 and 4 in Paper III.

The simulation results confirm the similarity in performance of Feedback TNN and the original TNN memory for sequence retrieval in either forward or backward directions. Both Cache-Winner and Explore-Winner reduce the random selections in the retrieval process and thereby the number of errors. This is reported as the number of cases in average that a retrieval algorithm picks the final component randomly from the candidate set. Therefore, in terms of achieving accurate sequence retrieval, both Cache-Winner and Explore-Winner retrievals are superior to the Winner, which randomly chooses from candidate set when the winner is not unique, and continues without further evaluations even when realizing a mistake in the previous retrieval steps. A comparison between retrieval error rate for different learning set sizes shows that retrieval performance with the Explore-Winner when using higher number of exploration steps is superior to the rest of scenarios. For higher density, Cache-Winner deals with larger cache memory which adds extra complexity. Similarly, Explore-Winner requires more computations

by exploring longer distances, and therefore finding an optimal exploration distance is a trade-off between time and accuracy. The simulation running time for different scenarios confirms that for reasonable density and consequently error values (retrieval error less than 0.1), the running time Cache-Winner and Explore-Winner remains at the same level of Winner. The source code for simulations in this paper is available at `https://github.com/Asieh-A-Mofrad/Tournament-Based-Sequence-Storage`.

Chapter 6

Future Research

Some of the research directions worth investigating in future studies are listed here:

- Both local-coding and precoding techniques that improve the clique-based model in Paper I can equally be used in the nested-clique model, Paper II, and tournament-based model, Paper III, to enhance the storage capacity and error tolerance in these structures.
- Parts of the motivation behind using self-dual additive codes over field $GF(4)$ in Paper I and nested-clique structures in Paper II, were to both benefit from the simple graph representation of these codes, and to develop neural-inspired networks that can be potentially significantly enhanced by overlaying with quantum graph state structures [see Danielsen, 2008, and references therein for self-dual additive codes and their graph representations]. Therefore, one direction could be to employ the graphical representation of these codes in the design of clique-based neural associative memories, and introduce more advanced (in comparison with Winner-Takes-All algorithm) message-passing algorithms for retrieval. Another direction worth investigating is whether a quantum associative memory [see, e.g., Ventura and Martinez, 2000] version of these (classic) associative memories is feasible. Note that the self-dual half-rate additive codes have directed graph representation [Danielsen and Parker, 2011] that might be interesting to use for sequences.
- In Paper II, the nested-clique structure can be extended to other types of nested structures such as Cartesian product of a clique (of clusters) and a cycle (of superclusters). For larger networks, the nested-clique structure can be extended to more layers, e.g. $K_{c_1}[K_{c_2}[K_{c_3}]]$. It is noteworthy that the nested-clique structure which is analogous to the Cartesian product of graphs is suggested partly for its

simplicity. Study of other nested-clique structures or other graph operations (say Inner product) could be an interesting research topic in its own right. Moreover, mathematical investigations of the optimal configuration of parameters could improve the design criteria. For instance, one could investigate the optimal number of nested layers (components in the graph multiplication) and the size of each cluster when either fixed amount of memory (number of connections) or fixed amount of neurons are available.

- The double-layered structure introduced by Jiang et al. [2016] combines a tournament-based hetro-association as the lower layer and a clique-based auto-associative memory as the upper layer. A hierarchical structure, similar to the double-layered structure, can be considered by adding an extra connectivity level to the Feedback TNN (Paper III). The mathematical part of this study also could be enhanced, for instance by finding the optimum distance of exploration for a specific density or retrieval error.

- It is possible to bridge between clique-based associative memories and some machine learning schemes with episodic memories that model the formation of equivalence relations in brain [Mofrad et al., 2020, 2021]. The problem of finding alternative stimulus configurations for which some attribute of a response remains invariant has been studied in many fields and under different names, including the field of psychology under the term stimulus equivalent [Sidman and Tailby, 1982; Stevens, 1951]. A stimulus equivalence class is formalized through relations in mathematical equivalence sets i.e. reflexivity ($A = A$), symmetry (if $A = B$ then $B = A$), and transitivity (if $A = B$ and $B = C$, then $A = C$). From another angle, formation of each equivalence class resembles construction of a clique between different categories and learning these associations nicely fit into those models which further can be used for modeling some language models based on relational frames [Berrou and Kim-Dufor, 2018; Hayes et al., 2021]. The most challenging part, however, might be to figure out the best way to use the binary clique-based neural network in the projective simulation agents' episodic memory [Briegel and De las Cuevas, 2012; Mautner et al., 2015] which has non-binary connections (uses Hebb's rule).

- Projective Simulation model is designed for both classic and quantum mechanics [see,e.g., Briegel and De las Cuevas, 2012; Paparo et al., 2014]. Further research can address quantum versions of self-dual codes and projective simulation in order to propose quantum associative memories [see, e.g., Ventura and Martinez, 2000] or more generally, quantum machine learning [for quantum machine learning see, Biamonte et al., 2017; Dunjko and Briegel, 2018, for instance].

Bibliography

A. Aboudib, V. Gripon, and X. Jiang. A study of retrieval algorithms of sparse messages in networks of neural cliques. In *COGNITIVE 2014: the 6th International Conference on Advanced Cognitive Technologies and Applications*, pages 140–146, 2014.

A. Aboudib, V. Gripon, and G. Coppin. A neural network model for solving the feature correspondence problem. In *International Conference on Artificial Neural Networks*, pages 439–446. Springer, 2016.

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

B. K. Aliabadi, C. Berrou, V. Gripon, and X. Jiang. Storing sparse messages in networks of neural cliques. *IEEE Transactions on neural networks and learning systems*, 25(5): 980–989, 2014.

C. Berrou and V. Gripon. Coded hopfield networks. In *Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on*, pages 1–5. IEEE, 2010.

C. Berrou and D.-H. Kim-Dufor. A connectionist model of reading with error correction properties. In *Human Language Technology. Challenges for Computer Science and Linguistics: 7th Language and Technology Conference, LTC 2015, Poznań, Poland, November 27-29, 2015, Revised Selected Papers*, volume 10930, page 304. Springer, 2018.

C. Berrou, O. Dufor, V. Gripon, and X. Jiang. Information, noise, coding, modulation: What about the brain? In *2014 8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pages 167–172. IEEE, 2014.

J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

J. A. Bondy, U. S. R. Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

J. Brea, W. Senn, and J.-P. Pfister. Sequence learning with hidden units in spiking neural networks. In *Advances in neural information processing systems*, pages 1422–1430, 2011.

H. J. Briegel and G. De las Cuevas. Projective simulation for artificial intelligence. *Scientific reports*, 2(1):1–16, 2012.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12 (Aug):2493–2537, 2011.

L. E. Danielsen. *On Connections Between Graphs, Codes, Quantum States, and Boolean Functions*. PhD thesis, The University of Bergen, 2008.

L. E. Danielsen and M. G. Parker. Directed graph representation of half-rate additive codes over gf(4). *Designs, Codes and Cryptography*, 59(1):119–130, 2011.

R. Danilo, H. Jarollahi, V. Gripon, P. Coussy, L. Conde-Canencia, and W. J. Gross. Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2501–2504. IEEE, 2015.

V. Dunjko and H. J. Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

R. Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

V. Gripon. *Networks of neural cliques*. PhD thesis, Télécom Bretagne, Université de Bretagne Occidentale, 2011.

V. Gripon and C. Berrou. Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, 22(7):1087–1096, 2011.

V. Gripon and C. Berrou. Nearly-optimal associative memories based on distributed constant weight codes. In *Information Theory and Applications Workshop (ITA), 2012*, pages 269–273. IEEE, 2012.

V. Gripon, J. Heusel, M. Löwe, and F. Vermet. A comparative study of sparse associative memories. *Journal of Statistical Physics*, 164(1):105–129, 2016.

G. B. Hacene, V. Gripon, N. Farrugia, M. Arzel, and M. Jezequel. Finding all matches in a database using binary neural networks. *COGNTIVE 2017*, page 67, 2017.

G. B. Hacene, V. Gripon, N. Farrugia, M. Arzel, and M. Jezequel. Budget restricted incremental learning with pre-trained convolutional neural networks and binary associative memories. *Journal of Signal Processing Systems*, 91(9):1063–1073, 2019.

J. Hawkins, D. George, and J. Niemasik. Sequence memory for prediction, inference and behaviour. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 364(1521):1203–1209, 2009.

S. C. Hayes, S. Law, K. Assemi, N. Falletta-Cowden, M. Shamblin, K. Burleigh, R. Olla, M. Forman, and P. Smith. Relating is an operant: A fly over of 35 years of rft research. *Perspectivas em Análise do Comportamento*, 2021.

D. O. Hebb. *The organization of behavior: a neuropsycholocigal theory*. New York, USA: Wiley & Sons, 1949.

H. Hoffmann. Sparse associative memory. *Neural computation*, 31(5):998–1014, 2019.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

J. J. Hopfield. Searching for memories, sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. *Neural Computation*, 20(5):1119–1164, 2008.

W. Imrich, S. Klavzar, and D. F. Rall. *Topics in graph theory: Graphs and their Cartesian product*. CRC Press, 2008.

H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross. Algorithm and architecture of fully-parallel associative memories based on sparse clustered networks. *Journal of Signal Processing Systems*, 76(3):235–247, 2014.

H. Jarollahi, V. Gripon, N. Onizawa, and W. J. Gross. Algorithm and architecture for a low-power content-addressable memory based on sparse clustered networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(4):642–653, 2015.

X. Jiang. *Storing sequences in binary neural networks with high efficiency*. PhD thesis, Télécom Bretagne, Université de Bretagne Occidentale, 2014.

X. Jiang, M. R. S. Marques, P.-J. Kirsch, and C. Berrou. Improved retrieval for challenging scenarios in clique-based neural networks. In *International Work-Conference on Artificial Neural Networks*, pages 400–414. Springer, 2015.

X. Jiang, V. Gripon, C. Berrou, and M. Rabbat. Storing sequences in binary tournament-based neural networks. *IEEE transactions on neural networks and learning systems*, 27(5):913–925, 2016.

T. N. Karaminis and M. S. C. Thomas. Connectionist theories of learning. In N. M. Seel, editor, *Encyclopedia of the Sciences of Learning*, pages 771–774. Springer US, Boston, MA, 2012.

D.-H. Kim, J. Park, and B. Kahng. Enhanced storage capacity with errors in scale-free hopfield neural networks: An analytical study. *PloS one*, 12(10):e0184683, 2017.

D. Krotov and J. J. Hopfield. Dense associative memory for pattern recognition. In *Advances in neural information processing systems*, pages 1172–1180, 2016.

B. Larras and A. Frappé. On the distribution of clique-based neural networks for edge ai. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4): 469–477, 2020.

B. Larras, P. Chollet, C. Lahuec, F. Seguin, and M. Arzel. A fully flexible circuit implementation of clique-based neural networks in 65-nm cmos. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(5):1704–1715, 2018.

D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

D. J. MacKay and R. M. Neal. Good codes based on very sparse matrices. In *IMA International Conference on Cryptography and Coding*, pages 100–111. Springer, 1995.

M. R. S. Marques, G. B. Hacene, C. E. R. K. Lassance, and P.-H. Horrein. Large-scale memory of sequences using binary sparse neural networks on gpu. In *2017 International Conference on High Performance Computing & Simulation (HPCS)*, pages 553–559. IEEE, 2017.

A. Maurer, M. Hersch, and A. G. Billard. Extended hopfield network for sequence learning: Application to gesture recognition. In *International Conference on Artificial Neural Networks*, pages 493–498. Springer, 2005.

J. Mautner, A. Makmal, D. Manzano, M. Tiersch, and H. J. Briegel. Projective simulation for classical learning agents: a comprehensive investigation. *New Gener. Comput.*, 33(1):69–114, 2015.

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

L. Medsker and L. C. Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

A. A. Mofrad, Z. Ferdosi, M. G. Parker, and M. H. Tadayon. Neural network associative memories with local coding. In *Information Theory (CWIT), 2015 IEEE 14th Canadian Workshop on*, pages 178–181. IEEE, 2015.

A. A. Mofrad, M. G. Parker, Z. Ferdosi, and M. H. Tadayon. Clique-based neural associative memories with local coding and precoding. *Neural computation*, 28(8): 1553–1573, 2016.

A. A. Mofrad, A. Yazidi, H. L. Hammer, and E. Arntzen. Equivalence projective simulation as a framework for modeling formation of stimulus equivalence classes. *Neural computation*, 32(5):912–968, 2020.

A. A. Mofrad, A. Yazidi, S. A. Mofrad, H. L. Hammer, and E. Arntzen. Enhanced equivalence projective simulation: A framework for modeling formation of stimulus equivalence classes. *Neural computation*, 33(2):483–527, 2021.

B. A. Olshausen and D. J. Field. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487, 2004.

G. D. Paparo, V. Dunjko, A. Makmal, M. A. Martin-Delgado, and H. J. Briegel. Quantum speedup for active learning agents. *Physical Review X*, 4(3):031002, 2014.

I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

G. J. Rinkus. A cortical sparse distributed coding model linking mini-and macrocolumn-scale functionality. *Frontiers in neuroanatomy*, 4:17, 2010.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

A. H. Salavati. *Coding theory and neural associative memories with exponential pattern retrieval capacity*. PhD thesis, EPFL, 2014.

A. H. Salavati and A. Karbasi. Multi-level error-resilient neural networks. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 1064–1068. IEEE, 2012.

A. H. Salavati, K. R. Kumar, A. Shokrollahi, and W. Gerstner. Neural pre-coding increases the pattern retrieval capacity of hopfield and bidirectional associative memories. In *2011 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 850–854. IEEE, 2011.

C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

M. Sidman and W. Tailby. Conditional discrimination vs. matching to sample: An expansion of the testing paradigm. *Journal of the Experimental Analysis of behavior*, 37(1):5–22, 1982.

F. T. Sommer. Associative memory and learning. In N. M. Seel, editor, *Encyclopedia of the Sciences of Learning*, pages 340–342. Springer US, Boston, MA, 2012.

F. T. Sommer and G. Palm. Improved bidirectional retrieval of sparse patterns stored by hebbian learning. *Neural Networks*, 12(2):281–297, 1999.

O. Sporns. Graph theory methods: applications in brain networks. *Dialogues in clinical neuroscience*, 20(2):111, 2018.

S. S. Stevens. *Handbook of experimental psychology.* Wiley, 1951.

D. Ventura and T. Martinez. Quantum associative memory. *Information Sciences*, 124 (1-4):273–296, 2000.

D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(5197):960–962, 1969.

Z. Yao, V. Gripon, and M. Rabbat. A gpu-based associative memory using sparse neural networks. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pages 688–692. IEEE, 2014.

Chapter 7

Scientific Results

Paper I

7.1 Clique-Based Neural Associative Memories with Local Coding and Precoding

Asieh Abolpour Mofrad, Matthew G. Parker, Zahra Ferdosi and Mohammad H. Tadayon

Mofrad, A. A., Parker, M. G., Ferdosi, Z., & Tadayon, M. H. (2016). Clique-based neural associative memories with local coding and precoding. Neural computation, 28(8), 1553-1573.

LETTER Communicated by Vincent Gripon

Clique-Based Neural Associative Memories with Local Coding and Precoding

Asieh Abolpour Mofrad
Asieh.Mofrad@uib.no
Matthew G. Parker
Matthew.Parker@ii.uib.no
Selmer Center, Department of Informatics, University of Bergen, Bergen 5020, Norway

Zahra Ferdosi
Ferdosi@aut.ac.ir
Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran

Mohammad H. Tadayon
Tadayon@itrc.ac.ir
Iran Telecommunication Research Center, Tehran 1439955471, Iran

Techniques from coding theory are able to improve the efficiency of neuroinspired and neural associative memories by forcing some construction and constraints on the network. In this letter, the approach is to embed coding techniques into neural associative memory in order to increase their performance in the presence of partial erasures. The motivation comes from recent work by Gripon, Berrou, and coauthors, which revisited Willshaw networks and presented a neural network with interacting neurons that partitioned into clusters. The model introduced stores patterns as small-size cliques that can be retrieved in spite of partial error. We focus on improving the success of retrieval by applying two techniques: doing a local coding in each cluster and then applying a precoding step. We use a slightly different decoding scheme, which is appropriate for partial erasures and converges faster. Although the ideas of local coding and precoding are not new, the way we apply them is different. Simulations show an increase in the pattern retrieval capacity for both techniques. Moreover, we use self-dual additive codes over field $GF(4)$, which have very interesting properties and a simple-graph representation.

1 Introduction

Neural associative memory is capable of memorizing (learning) a set of patterns and retrieving the full matching pattern from a given noisy fragment

Neural Computation **28, 1553–1573** (2016)
doi:10.1162/NECO_a_00856

of it. This functionality is similar to communication over a noisy channel. Channel coding concerns reliable and efficient retrieval of a set of patterns (code words in coding theory terminology) from a noisy version that the receiver receives. For digital error correcting codes, generally a subset of all possible pattern configurations is chosen for transmission. Coding techniques concern choosing this subset so that the receiver, which knows the allowed patterns (code words), can figure out whether the received pattern is an allowed one and, in the case of nonallowed patterns, finds the closest allowed pattern (i.e., decodes the received word to the most likely code word sent). Therefore, codes are carefully constructed to have high efficiency in the sense that the noise distance between pairs of code words is as large as possible given the size of the code set. On the other hand, neural associative memories are generally able to memorize any set of randomly chosen patterns, and as a consequence, they are not optimized for noise distance. Researchers who have applied coding theory to neural associative memories include Hopfield (2008), Berrou and Gripon (2010), Gripon (2011), Salavati (2014), and Berrou, Dufor, Gripon, and Jiang (2014). One approach in this context is to focus on learning patterns that have some sort of inherent redundancy; in another approach, the network is designed to be able to memorize any random set of patterns. For instance, Berrou and Gripon (2010) achieved considerable improvements in the pattern retrieval capacity of Hopfield networks by using Walsh-Hadamard sequences. Salavati, Kumar, Shokrollahi, and Gerstner (2011) proposed a neural association mechanism that employs binary neurons to memorize patterns belonging to another family of low-correlation sequences, called Gold sequences. These are discussed in Boguslawski, Gripon, Seguin, and Heitzmann (2014) and some strategies to store nonuniform patterns, such as by adding random bits and using Huffman coding, is a data compression technique.

Dividing a learned pattern into subpatterns can be shown to be useful in several ways (see Berrou & Gripon, 2010; Hopfield, 2008; Gripon & Berrou, 2011; Salavati & Karbasi, 2012; and for more details, Gripon, 2011, and Salavati, 2014). This approach also limits the allowed pattern configurations.

We follow the neural structure introduced in Gripon and Berrou (2011). These neural structures (called the GB model hereafter), which are based on Willshaw networks (Willshaw, Buneman, & Longuet-Higgins, 1969), are formed by dividing a neural network with n neurons into c clusters of size n/c each. The patterns are then chosen so that only one neuron in each cluster is active for a given pattern. Therefore, a pattern can be considered 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. To memorize a pattern, one then forms edges between active neurons and makes a clique (complete subgraphs) of order c. The decoding process is then to retrieve the erased nodes of the clique using edges stored during learning. It is worth mentioning that Hopfield (2008) developed a model of an associative memory

within a biological setting. In this model, neurons (n) are partitioned into a number of categories (say, c) with n/c possible values. A pattern then gets a single value in each category—the cluster counterpart to the GB model—and like the GB model, learning a new pattern is achieved by establishing edges between active neurons. Although the topology and learning part are similar, the retrieval part is different. There are other major differences that may be interesting to study because the Hopfield model focuses more on the biological aspects, whereas the GB model arises from coding techniques. For instance, for about the same number of neurons, the number of categories in the simulations is much larger than the number of clusters, and consequently the number of neurons in each Hopfield category is much less than in GB clusters. As an example, compare 50 categories, each with 20 neurons, versus 4 clusters, each with 256 neurons. In the Hopfield model, the pattern set is generated by randomly choosing a neuron in each category according to a power law distribution ($p(n) \sim \frac{1}{n^{1/2}}$), while in the GB model, active neurons in clusters are independent and identically distributed (i.i.d.). As mentioned previously, nonuniform distributions are also considered for the GB model (Boguslawski et al., 2014). Moreover, the Hamming distance between two patterns in the Hopfield model is defined as the number of neurons in which they differ, while in the GB model, the Hamming distance is the number of clique edges in which two patterns differ, which means that distance is far better for the latter case.

The GB-based models proposed in this letter focus on improving storage performance and making memory more resistant in the presence of partial erasure. Both local coding and precoding are techniques used to enhance pattern retrieval capacity and have been used in neural associative memory. For instance, clustering the neurons and applying the rule that just one neuron in each cluster is allowed to be active is itself a local coding (Hopfield, 2008; Gripon & Berrou, 2011). Another example is a two-level neural associative model in Salavati and Karbasi (2012) in which the pattern neurons are divided into clusters and each cluster is a bipartite graph—inspired from graph-based codes like LDPC (low-density parity check) codes, where subpatterns should form a subspace—a code in coding terminology. The second level may enforce constraints in the same subpattern space—just local coding—or in a totally different space—a combination of local coding and precoding.

The local coding construction proposed in this letter does not affect the number of neurons but adds redundancy to the patterns and then learns code words assigned to each subpattern in the neural network. Part of this work was presented at the 14th Canadian Workshop on Information Theory (Mofrad, Ferdosi, Parker, & Tadayon, 2015), and here we improve on the decoding algorithm introduced there to make it suitable for partial erasures. This reduces the size of the neurons involved in the retrieval process, and thus they converge faster, especially in the context of iterative decoding.

The precoding technique is a more straightforward way to improve the pattern retrieval capacity, and there is an argument that working with structured patterns is biologically meaningful and that sensory inputs to the brain are preprocessed before actually being stored (Salavati et al., 2011; Salavati, 2014; Berrou et al., 2014).

The precoding technique that we consider simply encodes the patterns and then splits the corresponding code words and memorizes each part in a cluster. We perform experiments in the presence of partial erasure and compare local coding and precoding models. These two schemes can then be combined if one needs more data protection.

For simulation, we select two error-correcting codes. The algebraic Reed-Solomon (RS) code is a maximum-distance-separable (MDS) linear code (MDS means that for a fixed code word length n and pattern length k, MDS codes have the greatest error-correcting and -detecting capabilities). RS codes are widely used for data storage and are suitable for erasure errors (Reed & Solomon, 1960).

The second class of error-correcting code that we select is the self-dual additive codes over $GF(4)$ (see Danielsen, 2008). These codes can be represented as simple graphs and have many interesting features. As far as we know, the graph-based codes that have been used in neuroinspired memories in the literature have bipartite representation (see Salavati, 2014, and Berrou & Gripon, 2010). In this letter, we do not consider the graphical representation of these codes; we just consider them as a second error-correcting code because these codes have more flexible parameters suited to the design of the network. In future work, we shall apply message-passing algorithms to the simple graphs representing these $GF(4)$-additive codes to improve decoding performance.

The rest of the letter is as follows. Section 2 reviews the basics and the clique-based networks introduced by Gripon and Berrou—notations from Gripon and Berrou (2011) mostly. Section 3 is devoted to the local coding scheme and precoding model. In section 4, we explain our decoding algorithm using an example. Section 5 contains the simulation results and a comparison of neural networks both with and without local coding. The results for local coding and precoding are also compared and discussed. Section 6 concludes. The detailed decoding algorithm is provided in the appendix.

2 GB Model of Neural Networks

Gripon and Berrou introduced a model where, by splitting a network of n neurons into c clusters of size $l = n/c$, any alphabet (say, $\mathcal{A}$) with cardinality $|\mathcal{A}| = l$ can be depicted. The model allows for different size alphabets and clusters, but for simplicity, it is considered fixed with $l = 2^\kappa$, so as to ease working with binary patterns. Each binary pattern of length

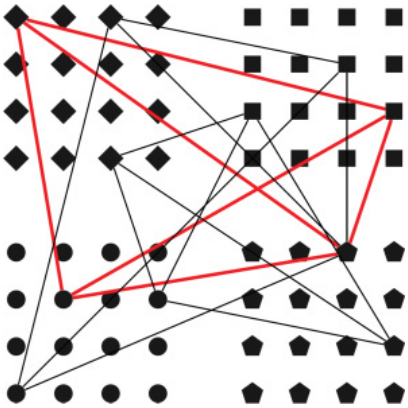


Figure 1: Learning process in a network with 64 neurons, which split into four clusters of 16 neurons each. Red edges represent the binary pattern 0000, 1011, 0101, 0010, which is learned as a clique.

κ is then assigned to a unique neuron or, equivalently, a character of alphabet $\mathcal{A}$,

$$f : \{0, 1\}^{\kappa} \to [|1; l|],$$

where $[|1; l|]$ is the subset of integers between 1 and l.

The learning process is simply to store patterns of length $k = c\kappa$ as cliques of size c where a unique neuron is selected from each individual cluster. More formally, consider learning pattern m,

$$C : m = m_1 m_2 \cdots m_c \to (f(m_1), f(m_2), \ldots, f(m_c)),$$

where each $m_i \in \{0, 1\}^{\kappa}$, $1 \leq i \leq c$ is a binary pattern of size κ. The active neurons, $f(m_i)$, connect together by edges to make a clique, as in Figure 1. The value of each neuron is considered binary; if a node is within a clique for a given pattern, its value is 1, and 0 otherwise. If $W(m)$ denotes the set of edges of the clique for pattern m, then the edges after learning a set of patterns, M, will be

$$W = \bigcup_{m \in M} W(m).$$

Retrieving or recalling part of a learned pattern is done in two steps and can be iterative. The algorithm finds the most probable active neuron in each cluster. Aboudib, Gripon, and Jiang (2014) and Jiang (2014) provide a detailed study of the retrieval algorithm. For instance, the different

approaches of global winners-take-all (GWsTA) and global losers-kicked-out (GLsKO) both improve the retrieval performance. Our decoding is designed for partial erasures and reduces computations.

3 Local Coding Technique and a Precoding Clique-Based Model

As mentioned in section 1, the GB model inherently has a local coding in which the allowed subpatterns are those with exactly a 1 in their binary representation (a kind of constant weight code in each cluster). However, our idea for local coding is to map a code word instead of a subpattern to each neuron. The English language is a good example to explain our local coding technique. Consider the set of learning patterns consisting of meaningful sentences with a fixed length (i.e., each with the same number of words—for instance, as a sample, this quote from Nelson Mandela as a pattern to be memorized: "A winner is a dreamer who never gives up." The network we choose has nine clusters, and there is a one-to-one map between all possible words (subpatterns) and the neurons in each cluster. A partial erasure then is like "A w-n–r i- - dr— w-o n–er giv-s up" and the local retrieval deals with the spelling of the words and meaningful words—well-separated code words in the model—and in the higher level, the grammar or the meaning of the sentence is checked—clique connections in the model. The local coding technique in this example is implemented in terms of the words that are allowed in the sentence; that is, codewords in the model are allowed words in this example.

Local codes can be chosen from different alphabets, rates, and minimum Hamming distances, and it is possible to consider different codes with different code book sizes for each cluster. The Hamming distance between two words of the same length, or code words, is the number of positions with different symbols. The minimum distance of a code[1] is the lowest Hamming distance between any two code words in the code. If we choose a code with high minimum distance and a partial erasure happens, then the minimum distance of a local code may eliminate that erasure and the ordinary decoding of GB neural networks can be done more efficiently.

More formally, consider that the goal is to learn patterns of type $m = m_1 m_2, \ldots, m_c$ where each m_i, $1 \leq i \leq c$ is a nonbinary pattern of size κ. Components of m_i can be binary as well, but we choose them from the finite field $GF(2^p)$ and use an algebraic Reed-Solomon (RS) code, which is a maximum-distance-separable (MDS) linear code and has the best possible minimum distance. Therefore, a neural network of c clusters, each with $l = 2^{p\kappa}$ neurons, can represent patterns like m. Recall that if no local coding is done, then each subpattern m_i maps to neuron $f(m_i)$ in the ith cluster,

[1] Minimum distance is a very important parameter in designing block codes.

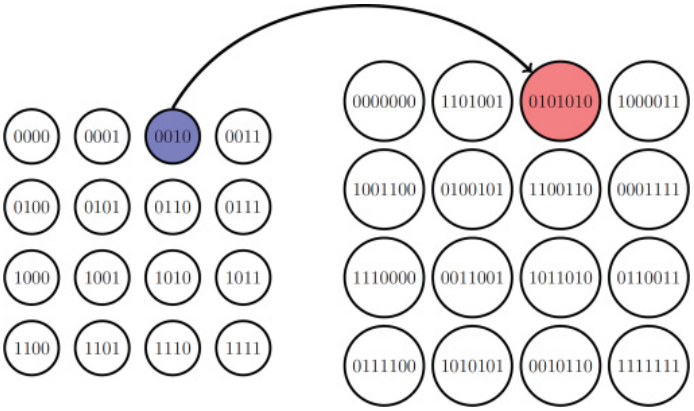


Figure 2: Each neuron in the cluster is assigned to a subpattern before local coding (left) and to a code word of Hamming code $(7, 4)$ after local coding (right).

$$m = m_1 m_2 \cdots m_c \to (f(m_1), f(m_2), \ldots, f(m_c)),$$

where $f : GF(2^p)^\kappa \to [|1; l|]$.

Linear codes, like RS codes, have a generator matrix whose rows form a basis for them. So code words of a code $\mathcal{C}$ with generator matrix G have code words like $g(m_i) = m_i G$ for each subpattern m_i. Then m maps to $m_g = g(m_1)g(m_2), \ldots, g(m_c)$ and $f : \mathcal{C} \to [|1; l|]$ maps a code word to a neuron and in general:

$$m = m_1 m_2, \ldots, m_c \to (f(g(m_1)), f(g(m_2)), \ldots, f(g(m_c))).$$

As a toy example let $l = 16$ and the local code be a Hamming code (7, 4) (Hamming, 1950); a binary subpattern $m_i = (m_i^1 m_i^2 m_i^3 m_i^4)$ is coded into $g(m_i) = (p_1 p_2 m_i^1 p_3 m_i^2 m_i^3 m_i^4)$ where $p_1 = m_i^1 + m_i^2 + m_i^4$, $p_2 = m_i^1 + m_i^3 + m_i^4$, and $p_3 = m_i^2 + m_i^3 + m_i^4$, where all additions are modulo 2. Then G is:

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \tag{3.1}$$

and it can be seen easily that $g(m_i) = m_i G$. See Figure 2 for the local coding scheme using Hamming code $(7, 4)$.

3.1 Precoding Clique-Based Neural Networks. We recruit another example from language to explain the precoding technique to make a

comparison between local coding and precoding easier. Consider the patterns after precoding to be a set of meaningful words—code words in the model—with the same number of syllables—the number of clusters in the network. A syllable, which may or may not have a meaning, is made up of phonemes, and each neuron represents a syllable. Because a phoneme is the smallest unit of sound that distinguishes one word from another, we can consider phonemes as the alphabet used in the precoding. For instance to memorize "astronomical," /æs.trə.nɑ.mɪ.kəl/, we need five clusters, and in each cluster we have a one-to-one map between all syllables and neurons.[2] On recall, a clue like /æ-.t-ə.--.m-.kəl/ may be given. Although there is not a meaning (a particular minimum distance), syllables are not a random combination of phonemes, and some degree of regularity holds that facilitates erasure correcting in clusters. The edges established to make cliques in this example can represent the spelling or meaning, for instance. In this example, the role of cliques is more important, and the distance between cliques is greater.

We compare local coding with precoding in section 5 by results from simulations.

4 Recalling from a Partially Erased Pattern

We receive a partially erased pattern from which erased symbols must be retrieved. Equivalently, in clique-based models, we must find a clique that contains the provided symbols as active neurons—neurons whose value is 1. As the given part of a learned pattern is assumed correct, then recalling is simply a matter of finding a match from memorized patterns. To avoid unnecessary computation, we introduce a decoding algorithm suitable for retrieving partial erasures. We explain the two-level retrieval algorithm by examples from the English language provided in section 3; a formal version of the algorithm can be found in the appendix.

Suppose that, from the partially erased sentence "A w-n— i- - dr—— w-o n–er giv-s up," the memory tries to recall the complete sentence. The first step is a local search within the clusters for all possible words that match. If there is a unique option, like "A" in the first cluster and "gives" "up" in the last but one cluster—we suppose there are no other words that match "giv-s up"—the corresponding neuron is active and all edges contained in the learned edge set W with one end point at these active neurons are established. Suppose for the second cluster there are candidate words (neurons): {'window,' 'winner,' 'winter,' 'winrar,' 'wonder'}, and also for the third cluster: {'id,' 'if,' 'in,' 'is,' 'it'}; fourth cluster: {'A,' 'I'};

[2]The length of syllables is not important in this example, but as the length is fixed in the model, one can consider a fixed 3 phoneme for all neurons and add an empty sign to those syllables with fewer phonemes.

fifth cluster: {'dragoon,' 'dreamed,' 'dreamer,' 'driving,' 'drunken'}; sixth cluster: {'who,' 'woo'}; and the seventh cluster: {'nagger,' 'nailer,' 'never,' 'number'}. A better minimum distance in local coding reduces the size of these candidate sets. The second level then checks the degree of each word. Starting from the second cluster, suppose the degree of 'wonder' is zero, the degree of 'window' and 'winrar' is 1, and the degree of 'winner' and 'winter' is 3. So from the sentence (cliques) information, we know that just two valid words remain at the second position (cluster). The word candidates are: {'winner','winter'}. By the same argument suppose new sets update as follows: third cluster: {'if,' 'is'}; fourth cluster: {'A'}; fifth cluster: {'dreamer,' 'drunken'}; sixth cluster: {'who'}; and the seventh cluster: {'never'}. So the active neurons in the fourth, sixth, and seventh clusters are found, and the algorithm repeats by establishing edges from these three neurons and checking the degree of each to remove the ones whose degree is less than 6. This recall may be successfully finished after one or two more iterations. But consider the case where we end up with the sentence, "A winner is a dr— who never gives up" and both remaining candidates, 'dreamer' and 'drunken,' have degree 8. In this case, recall fails. These kinds of failure would happen because there is no rule that forbids too similar sentences from being members of the learning set. More formally, subpatterns are chosen randomly, and although the clique form plays the role of grammar or meaning, for instance, sentences that are too close may still cause problems. In comparison, such a problem will not happen with the precoding technique because the learning set patterns have a high pairwise minimum distance.

Overall, a good local coding limits the possible matching set.[3] On the other hand, a precoding forces patterns to be well separated. The best strategy is to use both techniques together to have a more reliable memory.

For the last example, let the pattern set be all sentences of length c, with at least d_p different words between any pair of sentences, and at least d_l different letters between any two words in the same position. The condition is strict, but the greater d_p and d_p can be made, the more reliable the retrieval will be.

5 Simulation Results

To see the performance of the proposed associative memory with local coding, we first consider a network of 4096 neurons that are clustered in 8 sets, each with 512 neurons. For local coding, the $[7, 3, 5]_8$ RS code is used; that is, with this code, any subpattern of length 3 where its components are taken from $GF(8)$ maps to a code word of length 7 so that the minimum Hamming distance of the new set of code words is 5. By fixing the learning set size,

[3] For instance 'who' and 'woo' have Hamming distance 1, and in a good coding scheme, both cannot be code words simultaneously.

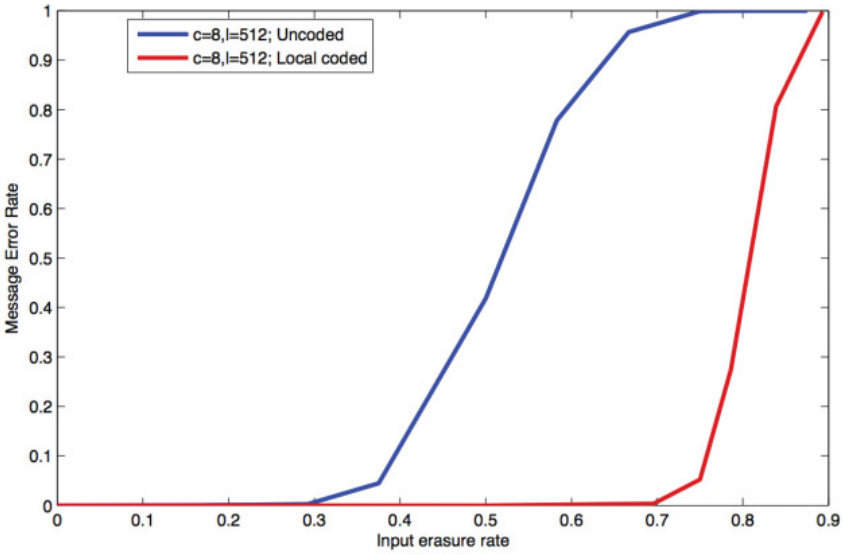


Figure 3: Comparison of performance of an uncoded associative memory of eight clusters, each with 512 neurons (blue curve), with the coded version (red curve) where local coding uses the $[7, 3, 5]_8$ RS code for $|M| = 50{,}000$.

we see the results for different partial erasure probabilities in Figure 3; the retrieval performance when erasure probability is fixed and learning set size is growing is shown in Figure 4.[4]

Figures 5 and 6 prosent the same comparison in a network of 512 neurons that are clustered in eight sets, each with 64 neurons when the local coding is the $(6, 2^6, 4)$ hexacode (see Conway & Sloane, 1988, for instance). The hexacode is a self-dual $GF(4)$ additive code and so can be represented by a simple graph. Its generator matrix corresponding to graph (b) is (see Figure 7):[5]

$$G = \begin{pmatrix} \omega & 1 & 1 & 1 & 0 & 0 \\ 1 & \omega & 1 & 0 & 1 & 0 \\ 1 & 1 & \omega & 0 & 0 & 1 \\ 1 & 0 & 0 & \omega & 1 & 1 \\ 0 & 1 & 0 & 1 & \omega & 1 \\ 0 & 0 & 1 & 1 & 1 & \omega \end{pmatrix}$$

[4]As can be seen, the performance obtained with the proposed local coding is dramatically better than the uncoded performance. The main cost of this performance is an increased word length (three symbols in $GF(8)$ to seven symbols in the same field) for each node.

[5]The generator matrix is obtained from the adjacency matrix of the graph b, by setting all the diagonal entries to ω.

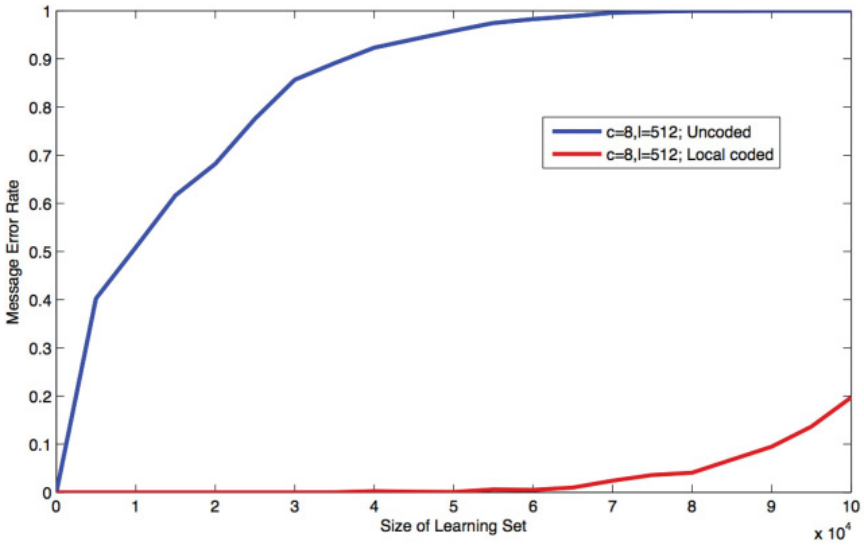


Figure 4: Comparison of performance for an uncoded associative memory of eight clusters, each with 512 neurons (blue curve), with the coded version where local coding uses the $[7, 3, 5]_8$ RS code (red curve). The erasure rate for each symbol is 0.7. The largest data set here is 100,000.

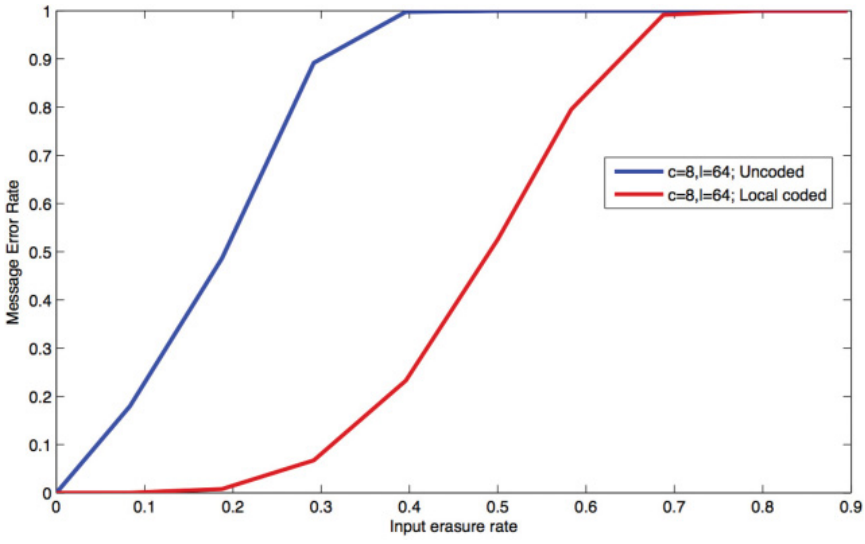


Figure 5: Comparison of performance of an uncoded associative memory of eight clusters, each with 64 neurons (blue curve), with the coded version where local coding uses the hexacode $(6, 2^6, 4)$ for $|M| = 4000$.

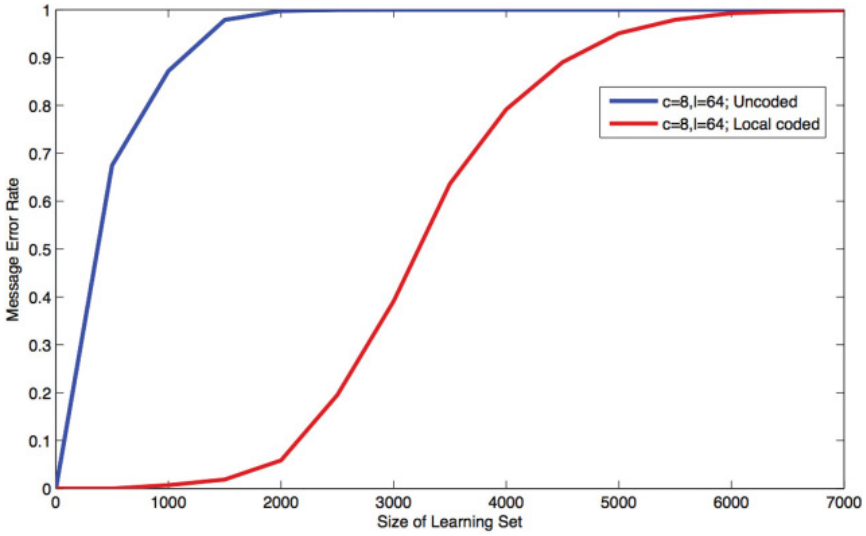


Figure 6: Comparison of performance for an uncoded associative memory of eight clusters, each with 64 neurons (blue curve), with the coded version where local coding uses the hexacode $(6, 2^6, 4)$. The erasure rate for each symbol is 0.6.

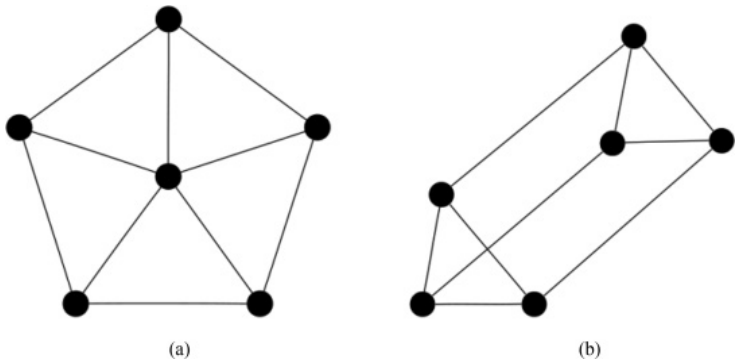


Figure 7: Two graph representations of the hexacode (Danielsen, 2008).

As opposed to the RS code, the number of neurons using local coding with the hexacode does not change. The length of each subpattern remains fixed, but the field changes from $GF(2)$ to $GF(4)$.[6]

[6]The hexacode is an additive code: a binary vector m_i (subpattern for local coding) generates all 2^6 code words by m_iG (i.e., m_i is taken over $GF(2)$, not $GF(4)$).

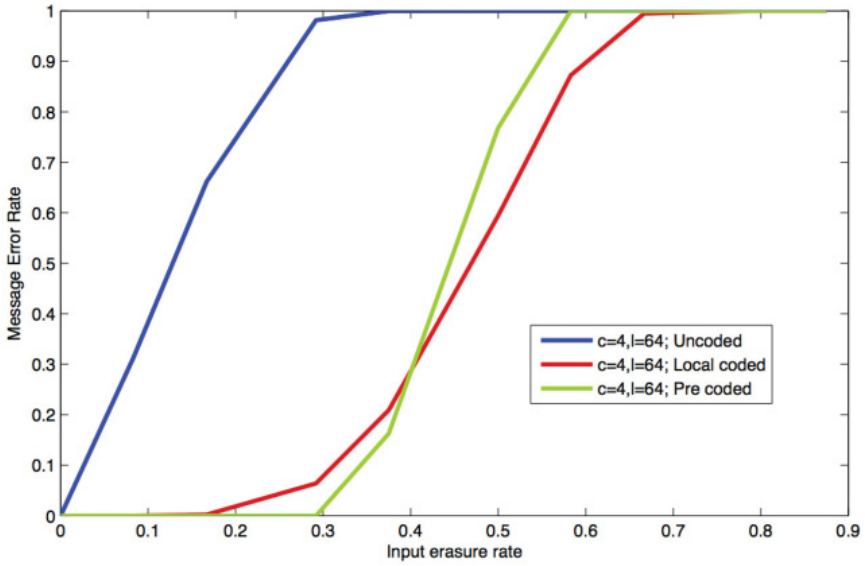


Figure 8: Comparison of performance of an uncoded associative memory of four clusters, each with 64 neurons (blue curve), with the local coded version where local coding uses the hexacode $(6, 2^6, 4)$ (red curve) and is precoded with the $(12, 2^{12}, 6)$ code (green curve) for $|M| = 3000$.

As mentioned in section 1, we propose using self-dual $GF(4)$ additive codes because their parameters are more flexible. Moreover, we also intend to use such codes in our future work because of their graphical representations. In particular, it is known that nested-clique graphs represent many of the strongest $GF(4)$ additive codes in terms of pairwise distance and optimum edge sparsity and are therefore good candidates for using to build nested-clique neural networks. The idea would be to embed a self-dual code inside each neuron and benefit from this graph code during the retrieval process.

For the precoding technique, we choose a $(12, 2^{12}, 6)$ self-dual $GF(4)$ additive code (the dodecacode) (Calderbank, Rains, Shor, & Sloane, 1998). This code maps any binary pattern of size 12 to a code word of size 12 in $GF(4)$ with minimum distance 6. We have $c = 4$ clusters, each with $l = 64$ neurons, and the length of each subpattern is 3. This is compared to an uncoded version as well as to a local coding version. For the local coding, we use the hexacode again, but the number of clusters is set to be 4.

Figures 8 and 9 confirm the expectation that precoding improves the capability of memorizing a larger set of patterns and recalling successfully in the presence of stronger partial erasure. From Figure 8, we see that when the erasure rate is smaller than 0.4, the precoding technique gives better results. But in the case of higher erasure probability, local coding outperforms. This

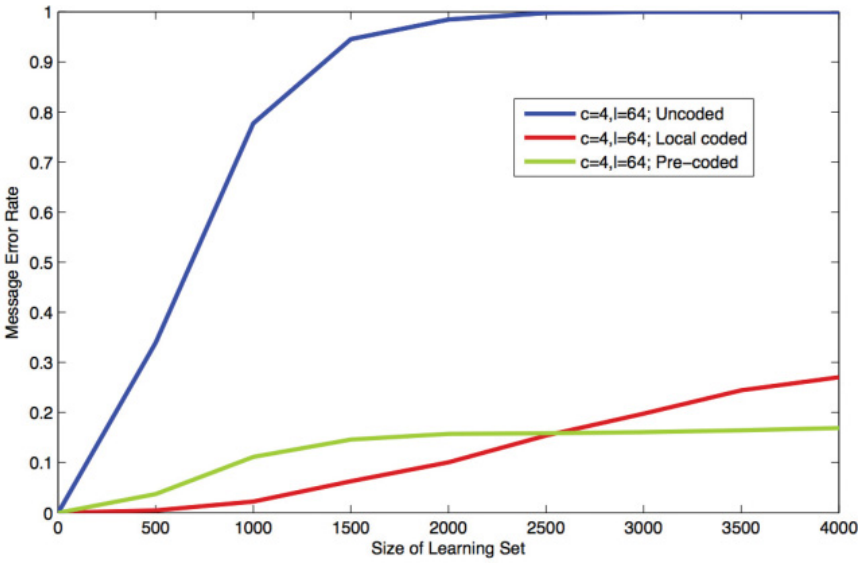


Figure 9: A comparison of the performance for an uncoded associative memory of four clusters, each with 64 neurons (blue curve), with the local coded version where local coding uses the hexacode $(6, 2^6, 4)$ (red curve) and precoded with the $(12, 2^{12}, 6)$ code (green curve). The erasure rate for each symbol is 0.4.

is justified by the following argument: after the first check, whenever partial erasure is low, the number of clusters with an active neuron is large. So the precoding technique is able to benefit more from its minimum distance than that associated with the distance between cliques. On the other hand, when erasure probability is higher, the ability of local retrieval is more important, which is what the local coding model was designed for.

Then to compare storage capacity, we fix erasure at 0.4, where both techniques showed a similar error rate in retrieval. Figure 9 shows that when the learning set is smaller, the local coding technique performs better, while for larger learning sets, the precoding technique outperforms. Again, this result is as expected. For a fairly small learning set the number of edges is smaller, and the cliques are more likely to have a higher minimum distance, so the role of local coding is more important. In contrast, when the number of edges is increased by increasing the size of the learning set, the role of precoding and minimum distance among the cliques becomes more important.

Note that in all simulations, the learning part is done independent of whether local coding is done. Indeed the symbols in the patterns are considered independent and identically distributed random variables. If we do

local coding, the set of edges is exactly the same, but precoding changes the shape of cliques, so W is no longer the same.

As the data set is chosen randomly, we repeat the experiment 100 times and compute the average to have more reliable results (i.e., choose 100 random patterns from the learning set as the input and partially erase it). Again, for any erasure probability, the symbols to be erased are chosen randomly, so we partially erased each pattern by 100 different randomly chosen erasure vectors. We then tried retrieving the chosen pattern, and if the pattern is completely retrieved, the algorithm is successful; otherwise, it fails. Finally, the ratio can be computed for each data set and an average taken over 100 different erasure vectors.

6 Conclusion

Some applications of coding techniques in neuroinspired associative memories have been discussed, and we have shown that both the local coding and precoding models based on the GB model have excellent error performance in the presence of partial erasures. The results are somewhat theoretical, but due to their structure and ability to retrieve patterns from a partial clue, such memories have potential application to content-addressable memories and search engine algorithms. Our simulation results suggest that the local coding model is better suited to the case where the erasure probability is high or the learning set is quite small. In contrast, the precoding model seems to be more suited to the situation where the erasure probability is not that high or the size of the learning set is rather large. We present a new version of the decoding algorithm, which reduces the computational complexity and is suitable for partial erasures.

It is not necessary that the local coding be nonbinary or use extension fields over $GF(2^p)$. We chose RS codes because they are suitable for storage and erasure-type errors. We also considered self-dual $GF(4)$ additive codes, as we shall exploit their graph representations in future work.

Appendix: Detailed Version of the Recalling Algorithm for a Partially Erased Pattern

The detailed version of the recalling algorithm is provided here. It assumes the neural network has local coding, but it can also be used for an uncoded version, similar to a precoding model as well. We assume that the same code $\mathcal{C}$ is used in all clusters.

Consider that a noisy version of a learned pattern,

$$m_g = \left(m_{1_1} m_{1_2} \cdots m_{1_t} \middle| m_{2_1} m_{2_2} \cdots m_{2_t} \middle| \cdots \middle| m_{c_1} m_{c_2} \cdots m_{c_t}\right),$$

is $\hat{m}_g = (\hat{m}_{1_1}\hat{m}_{1_2}\cdots\hat{m}_{1_t}|\hat{m}_{2_1}\hat{m}_{2_2}\cdots\hat{m}_{2_t}|\cdots|\hat{m}_{c_1}\hat{m}_{c_2}\cdots\hat{m}_{c_t})$, where symbol m_{i_r} is given—$\hat{m}_{i_r} = m_{i_r}$—or is erased—$\hat{m}_{i_r} = e$—and t is the length of the code word, so $t > \kappa$ if we have local coding and $t = \kappa$ otherwise.

We also assign $\hat{m}_i = m_i$ iff all the m_{i_r} are known and $\hat{m}_i = e$ iff at least, for one r, $\hat{m}_{i_r} = e$.

To begin, we separate clusters into two sets, C_u and C_e, for unerrored and errored components, respectively:

$$C_u = \{i : \hat{m}_i = m_i\} \text{ and } C_e = \{i : \hat{m}_i = e\}, 1 \le i \le c.$$

Each neuron is shown with $n_{ij}, 1 \le i \le n, 1 \le j \le l$, which is equivalent to a unique value in $\{0, 1, 2, \ldots, l-1\}$. So n_{ij} is a node in the graphical representation of m_i. For a specific j in cluster i, we assign $n_{ij} = f(m_i)$. Note that m_i is a code word here.

Also for $i \in C_e$, we define and initialize sets $T(i) = \{n_{ij} | \hat{m}_{i_r} = m_{i_r} \text{ or } \hat{m}_{i_r} = e; \forall j, r\}$. As we will see, these sets play an important role in reducing computational complexity.

Once we construct all $T(i)$ sets, a local check is done as follows. For all $i \in C_e$, if $T(i) = \{n_{ij}\}$ for some j, then

- Let $C_u = C_u \cup \{i\}$ and $C_e = C_e \setminus \{i\}$.
- Correct $\hat{m}_i$ by putting $\hat{m}_i = f^{-1}(n_{ij})$.

Indeed as the n_{ij} correspond to code words of a code, some erasures were corrected by this local check.

Values of the neurons are defined as

$$v(n_{ij}) = \begin{cases} 1 & \text{if } i \in C_u \text{ and } n_{ij} = f(m_i), \\ 0 & \text{otherwise.} \end{cases} \tag{A.1}$$

We establish all edges contained in the learned edge set W, so that at least one node for each edge has value 1. More formally, we initialize the edge set $w = \{(n_{ij}, n_{i'j'}) \in W | v(n_{ij}) = 1 \text{ or } v(n_{i'j'}) = 1\}$, where $1 \le i, i' \le c, 1 \le j, j' \le l$.

Now we can start iterative retrieval (see algorithm 1). If the first part of the algorithm (until line 32) retrieves the original pattern, it stops; if there are several candidate neurons in each cluster C_e, we search in W for edges with end nodes in the $T(i)$ that make a clique of size $|C_e|$. This happens when the partial erasure is high and distributed so the active neuron in most clusters is unknown. Note that the definition of w is changed for the second part of the recalling (line 34).

Algorithm 1: Retrieval Algorithm.

Require: Initialize C_e, C_u, $T(i)$ for $i \in C_e$, $d(n_{ij}) = deg(n_{ij})$ for all neurons in $T(i)$, $v(n_{ij})$, w, $v_{\max} = |C_u|$, $Flag = True$, $Counter = 0$, $Retrieval = Failed$

1: **while** $Flag = True$ **do** ▷ In this loop the algorithm removes elements from the $T(i)$ sets and finds an active neuron in cluster i whenever $|T(i)| = 1$

2: **if** $v_{\max} = c$ **then** ▷ $v_{\max} = c$ means all clusters have their unique active neuron, Algorithm stops by setting $Flag = False$

3: $Retrieval = Succeed$

4: $Flag = False$

5: **return** $Retrieval$

6: **else if** $v_{\max} = 0$ **then** ▷ $v_{\max} = 0$ means no active neuron is found at the first stage; go to line 33

7: $Flag = False$

8: **else**

9: **for** $i \in C_e$ **do** ▷ The edges cause some candidates to be removed from $T(i)$

10: **for** $n_{ij} \in T(i)$ **do**

11: **if** $d(n_{ij}) \neq v_{\max}$ **then**

12: $T(i) \leftarrow T(i) \setminus \{n_{ij}\}$ ▷ i.e., a new active neuron is found

13: **end if**

14: **end for**

15: **if** $|T(i)| = 1$ (i.e., $T(i) = \{n_{ij}\}$ for one j) **then**

16: $v(n_{ij}) = 1$

17: $\hat{m}_i = f^{-1}(n_{ij})$

18: $w = w \cup \{(n_{ij}, n_{i'j'}) \in W\}$ ▷ New edges establish from new activated neurons; C_u and C_v will update

19: $C_u = C_u \cup \{i\}$

20: $C_e = C_e \setminus \{i\}$

21: $Counter = Counter + 1$

22: **end if**

23: **end for**

24: **end if**

25: **if** $Counter \geq 1$ **then** ▷ i e., a new active neuron is found in this iteration

26: Update $d(n_{ij})$ based on updated w

27: $v_{\max} = |C_u|$

28: $Counter = 0$

29: **else**

30: $Flag = False$ ▷ i.e., exit while loop and go to line 33

31: **end if**

32: **end while**

33: **if** $v_{\max} < c$ **then** ▷ Now all the remaining $T(i)$ sets have more than one neuron, so the algorithm searches for a clique among neurons in $T(i)$ sets by establishing edges with at least one end within candidates

34: $w = \{(n_{ij}, n_{i'j'}) \in W | n_{ij} \in T(i) \text{ or } n_{i'j'} \in T(i')\}$

35: where $i, i' \in C_e, 1 \leq j, j' \leq l$

36: $N_{i' \to i} = \{n_{ij} | (n_{ij}, n_{i'j'}) \in w, 1 \leq j, j' \leq l\}$

37: where $i \neq i'$ and $i, i' \in C_e$

38: $d(n_{ij}) = \sum_{i' \in C_e \setminus \{i\}} \chi_{N_{i' \to i}}(n_{ij})$ ▷ The characteristic function $\chi_{N_{i' \to i}}$ is used to show the number of connected clusters to n_{ij} by at least one candidate neuron. The number of connections from a specific cluster does not matter.

39: $Flag = True$

40: **while** Flag=True **do**

41: **if** $v_{\max} = c$ **then**

42: $Retrieval = Succeed$

43: $Flag = False$

44: **else**

45: **for** $i \in C_e$ **do**

46: **for** $n_{ij} \in T(i)$ **do**

47: **if** $d(n_{ij}) \neq |C_e| - 1$ **then** ▷ Those neurons that are not connected to all clusters in C_e are removed from $T(i)$

48: $T(i) \leftarrow T(i) \setminus \{n_{ij}\}$

49: **end if**

50: **end for**

51: **if** $|T(i)| = 1$ (i.e. $T(i) = \{n_{ij}\}$) **then** ▷ An active neuron n_{ij} is found

52: $v(n_{ij}) = 1$

53: $\hat{m}_i = f^{-1}(n_{ij})$

54: $w = w \setminus \{(n_{ij''}, n_{i'j'}) \in w | 1 \leq i' \leq c \text{ and } 1 \leq j', j'' \leq l\}$

▷ Updating w by removing edges from the newly activated neuron n_{ij}

55: $C_u = C_u \cup \{i\}$

56: $C_e = C_e \setminus \{i\}$

57: Update $N_{i \to i'}$ by new C_e and w sets

58: $Counter = Counter + 1$

59: **end if**

60: **end for**

61: **if** $Counter \geq 1$ **then** ▷ At least one neuron is activated

62: Update $d(n_{ij})$ based on updated $N_{i \to i'}$

63: $v_{\max} = |C_u|$

$Counter = 0$

else

$Flag = False$ ▷ No new neuron is activated in the last iteration

end if

end if

end while

end if

return $Retrieval$

References

Aboudib, A., Gripon, V., & Jiang, X. (2014). A study of retrieval algorithms of sparse messages in networks of neural cliques. In *Proceedings of COGNITIVE 2014: The 6th International Conference on Advanced Cognitive Technologies and Applications* (pp. 140–146). N.p.: IARIA.

Berrou, C., Dufor, O., Gripon, V., & Jiang, X. (2014). Information, noise, coding, modulation: What about the brain? In *Proceedings of the 2014 8th International Symposium on Turbo Codes and Iterative Information Processing* (pp. 167–172). Piscataway, NJ: IEEE.

Berrou, C., & Gripon, V. (2010). Coded Hopfield networks. In *Proceedings of the 2010 6th International Symposium on Turbo Codes and Iterative Information Processing* (pp. 1–5). Piscataway, NJ: IEEE.

Boguslawski, B., Gripon, V., Seguin, F., & Heitzmann, F. (2014). Huffman coding for storing non-uniformly distributed messages in networks of neural cliques. In *Proceedings of AAAI 2014: The 28th Conference on Artificial Intelligence* (vol. 1, pp. 262–268). Palo Alto, CA: AAAI.

Calderbank, A., Rains, E., Shor, P., & Sloane, N. (1998). Quantum error correction via codes over gf (4). *IEEE Transactions on Information Theory*, *44*(4), 1369–1387.

Conway, J. H., & Sloane, N. J. (1988). *Sphere packings, lattices and groups*. New York: Springer.

Danielsen, L. E. (2008). *On connections between graphs, codes, quantum states, and boolean functions*. PhD diss., University of Bergen.

Gripon, V. (2011). *Networks of neural cliques*. PhD diss., Télécom Bretagne.

Gripon, V., & Berrou, C. (2011). Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, *22*(7), 1087–1096.

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, *29*(2), 147–160.

Hopfield, J. J. (2008). Searching for memories, sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. *Neural Computation*, *20*(5), 1119–1164.

Jiang, X. (2014). *Storing sequences in binary neural networks with high efficiency*. PhD diss., Télécom Bretagne, Université de Bretagne Occidentale.

Mofrad, A. A., Ferdosi, Z., Parker, M. G., & Tadayon, M. H. (2015). Neural network associative memories with local coding. In *Proceedings of the 2015 IEEE 14th Canadian Workshop on Information Theory* (pp. 178–181). Piscataway, NJ: IEEE.

Reed, I. S., & Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, *8*(2), 300–304.

Salavati, A. H. (2014). *Coding theory and neural associative memories with exponential pattern retrieval capacity*. PhD diss., École polytechnique fédérale de Lausanne.

Salavati, A. H., & Karbasi, A. (2012). Multi-level error-resilient neural networks. In *Proceedings of the 2012 IEEE International Symposium on Information Theory* (pp. 1064–1068). Piscataway, NJ: IEEE.

Salavati, A. H., Kumar, K. R., Shokrollahi, A., & Gerstner, W. (2011). Neural precoding increases the pattern retrieval capacity of Hopfield and bidirectional associative memories. In *Proceedings of the 2011 IEEE International Symposium on Information Theory* (pp. 850–854). Piscataway, NJ: IEEE.

Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature*, *222*, 960–962.

Received November 21, 2015; accepted March 20, 2016.

Paper II

7.2 Nested-Clique Network Model of Neural Associative Memory

Asieh Abolpour Mofrad and Matthew G. Parker

Mofrad, A. A., & Parker, M. G. (2017). Nested-clique network model of neural associative memory. Neural Computation, 29(6), 1681-1695.

LETTER Communicated by Vincent Gripon

Nested-Clique Network Model of Neural Associative Memory

Asieh Abolpour Mofrad
Asieh.Mofrad@uib.no
Matthew G. Parker
Matthew.Parker@ii.uib.no
Selmer Center, Department of Informatics, University of Bergen, Bergen 5020, Norway

Clique-based neural associative memories introduced by Gripon and Berrou (GB), have been shown to have good performance, and in our previous work we improved the learning capacity and retrieval rate by local coding and precoding in the presence of partial erasures. We now take a step forward and consider nested-clique graph structures for the network. The GB model stores patterns as small cliques, and we here replace these by nested cliques. Simulation results show that the nested-clique structure enhances the clique-based model.

1 Introduction

There has been much recent activity regarding the design of sparse neural networks in general and neuro-inspired associative memories in particular. Sparse neural networks may refer to several categories. Gripon, Heusel, Löwe, and Vermet (2015) studied models of associative memories with sparse information, that is, patterns in the learning set are random strings of 0s and 1s with about $\log n$ 1s, only. We built our model on top of the clique-based model of Gripon, Berrou, and coauthors (GB model; Gripon & Berrou, 2011), where the assumption of the GB model is that for n neurons, there are c clusters of neurons with $1 \leq c \leq \log n$ and each pattern in the learning set has exactly one active neuron per cluster.[1] We have obtained very good results by applying precoding and local coding schemes on top of the GB model (Mofrad, Parker, Ferdosi, & Tadayon, 2016).

Hopfield (2008) discussed a similar cluster-based network but from a biological perspective. In this model, neurons (n) are partitioned into a number of categories (say, c)—the cluster counterpart to the GB model—with n/c possible values. A pattern activates a single neuron in each category, and like the GB model, learning the pattern is achieved by establishing edges between the activated neurons. The number of neurons in each Hopfield

[1] In graph theory, a clique defines as a complete subgraph.

Neural Computation **29, 1681–1695** (2017)
doi:10.1162/NECO_a_00964

category is much less than in GB clusters for about the same number of neurons. As an example, compare 50 categories, each with 20 neurons versus 4 clusters, each with 256 neurons.

Although the topology and learning rule are similar, the retrieval part and learning set distribution are different. In the Hopfield model, the pattern set is generated by randomly choosing a neuron in each category, according to a power law distribution ($p(n) \sim \frac{1}{n^{1/2}}$), while in the GB model, active neurons in clusters are independent and identically distributed (i.i.d.). Moreover, the Hamming distance between two patterns in the Hopfield model is defined as the number of neurons in which they differ, while in the GB model, the Hamming distance is the number of edges in which two patterns differ, which means that distance is far better for the latter case. A precise and detailed comparison of these two models might be an interesting issue to study; however, it goes beyond the framework of this letter.

Sparse patterns lead to sparse network connections, and much research has been done on sparse networks where the patterns are chosen randomly. In these models, synaptic connections exist only between neighboring neurons. Random dilution of connections in Hopfield networks has been studied in detail (see Gardner, 1989, for instance). However, cortical connectivity is better modeled by networks consisting of several modules with dense internal connections and sparse intermodular connections, so much attention has been given to the Hopfield model in the context of small-world (Bohland & Minai, 2001) and scale-free (Stauffer, Aharony, da Fontoura Costa, & Adler, 2003) models (see also Hilgetag & Goulas, 2016, for an investigation about the human brain from the network perspective). The GB model is a sparse model, but to the best of our knowledge, it does not yet fit into this model of dense internal connections and sparse intermodular connections. The nested-clique model proposed in this letter benefits from the idea of a single active neuron in the subclusters, with dense internal connections between active neurons in a cluster and sparse connections between clusters.

On top of the GB model, several extensions has been done, including further sparse organization (Aliabadi, Berrou, Gripon, & Jiang, 2014); that is, a sparse pattern is mapped to a unique neuron of a smaller set of clusters. Replacing nonoriented connections with directed ones in a way that the network can store sequential information in a tournament-based neural network (Jiang, Gripon, Berrou, & Rabbat, 2016) is another extension.[2] A double-layered structure introduced in Jiang et al. (2016) is a good combination of a tournament-based hetroassociation as the lower layer and an upper layer in the form of clique-based autoassociative similar to the sparse networks. The nested-clique model can be considered as a two-layer

[2]In graph theory, a tournament is a directed graph obtained by assigning a direction for each edge in a complete (sub)graph

network as well, where each layer is a clique-based autoassociative network and layers intertwine with each other.

Although not pursued in this letter, part of the motivation for this work is to develop neural-inspired networks that can be potentially significantly enhanced by overlaying with quantum networks that exploit quantum nonlocality and entanglement. Moreover, since the best zero-dimensional quantum codes often appear to have a nested-clique structure that is also optimally edge sparse (Danielsen & Parker, 2004), it would then be possible to overlay the results of nested-clique associative memories with quantum graph state structures so as to enhance performance by exploiting nonlocality, superposition, and entanglement.[3]

The rest of the letter is as follow. Section 2 reviews the basics and the clique-based networks introduced by Gripon and Berrou using mostly notations from Gripon and Berrou (2011). Section 3 is devoted to the nested-clique scheme and a brief review and comparison of the local coding and precoding model. Section 4 contains the simulation results and a comparison of clique-based versus nested-clique-based networks. Section 5 concludes.

2 Clique-Based Model of Associative Memory

In the design of neural networks, a neuron is a mathematical function that models a biological neuron that receives a weighted input sum and computes its output state by a nonlinear transform. The main task of neural association is to choose the graph weights w_{ij} so that the network is able to memorize M binary patterns of length k. We are mostly interested in autoassociation: retrieving a memorized pattern from its noisy version. For a neural associative memory design, we have to determine the topology of the neural network, the learning process (updating weights between neurons), and the recalling algorithm.

In the model that Gripon and Berrou introduced, by splitting the network of n neurons into c clusters of size $l = n/c$—supposing l is a power of 2 and $\kappa = \log_2(l)$—any alphabet (say, $\mathcal{A}$) with cardinality $|\mathcal{A}| = l$ can be depicted by neurons in each cluster. Each binary pattern of length $k = c\kappa$ is then assigned to a unique set of neurons or, equivalently, to a set of characters of alphabet $\mathcal{A}$:

$$m = m_1 m_2 \cdots m_c \to (f(m_1), f(m_2), \ldots, f(m_c))$$
$$\text{where} \quad f : \{0,1\}^\kappa \to \mathcal{A}.$$

[3] However, it remains to find and construct such nested-clique structures for larger graphs, an interesting problem in graph theory, that is, to what extent they exist as the number of graph nodes grows.

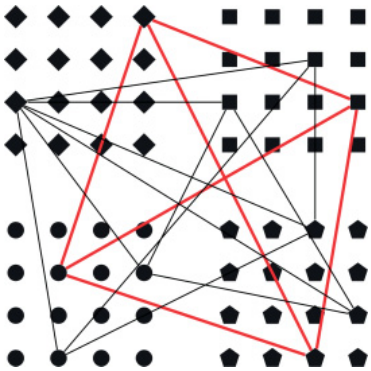


Figure 1: Learning process in a network with 64 neurons, split into four clusters of 16 neurons each. Red edges represent the binary pattern $m = (0011, 1011, 0101, 1110)$, which is learned as a clique.

The learning process is simply to connect the selected neurons together by edges to make a clique. The value of each neuron is considered binary; if a neuron is within a clique for a given pattern, its value is set to 1 and set to 0 otherwise. The weights of the edges in the network are all 1, and the learning process provides a set of edges W. This learning type (based on the Willshaw model) does not depend on the number of patterns that use neurons i and j, but on whether there is any pattern m^μ with $m_i^\mu m_j^\mu = 1$ (clipped synapses) (see Figure 1 as an example).

The recall phase or retrieving part of a partially erased learned pattern, $\hat{m}$, involves finding the most probable active neuron in each cluster and in general finding a match from the memorized patterns. Equivalently, in clique-based models, we must find a clique that contains the provided symbols as active neurons—neurons whose value is 1. Since we assume partial erasure-type error, the retrieval algorithm is slightly different from algorithms provided for GB neural networks and its variants. The detailed version of algorithm is provided in Mofrad et al. (2016) as an appendix.

The recall procedure, like general clique-based neural network models, consists of two steps: the global dynamics that establishes or eliminates connections based on provided information from erased pattern, $\hat{m}$, and a local decision that has been made to activate neurons. The winner-take-all rule activates neurons with the highest activity (or maximum degree) while the losers kicked-out rule (LsKO) eliminates active neurons with less activity using a threshold filter (see Jiang, 2014).

The modified version of the algorithm we used introduces a candidate set, $T(i)$, in each cluster, i, where partial erasure happens. The neurons in a candidate set will not be activated as long as their chances for being the final unique active neuron are equal. These candidate sets will be updated through retrieval by kicking out neurons that, based on dynamics of the

Algorithm 1: Retrieval Algorithm.

Data: $\hat{m}$, W

Result: The retrieved pattern m or failure message

begin

Based $\hat{m}$, each subpattern $\hat{m}_i$ maps to a unique active neuron n_{ij} or to a candidate set $T(i)$. /* The former occurs if $\hat{m}_i = m_i$ and the latter in the case partial erasure occurs. */

All edges from W with at least one end as an active neuron are established.

while *$\hat{m}$ not retrieved and at least a $|T(i)| = 1$ found in the last iteration* **do**

for *All $T(i)$* **do**

Find the max degree of $n_{ij} \in T(i)$: $v^i_{\max}$. The neurons with degree equal $v^i_{\max}$ remains in the set and the rest will be kicked out.

if $|T(i)| = 1$ **then** /* i.e., $T(i)$ has a unique neuron n_{ij} */

n_{ij} is activated, edges in W with one end as n_{ij} are established.

$T(i)$ is removed.

if *Cluster i with $|T(i)| \geq 2$ exists* **then** /* The active neurons provide no more information for making decisions */

Put aside clusters with active neurons /* and therefore all established edges at previous step */

Edges with at least one end in the remained candidate sets are established.

while *$\hat{m}$ not retrieved and at least a $|T(i)| = 1$ found in the last iteration* **do**

for *All $T(i)$* **do**

*nested **if** *n_{ij} does not connect to all other remaining candidate sets with at least one edge* **then**

n_{ij} is kicked out from $T(i)$. /* By kicking out n_{ij} from $T(i)$, all the connected edges are removed */

if $|T(i)| = 1$ **then**

$n_{ij} \in T(i)$ is activated, $T(i)$ is removed.

network, cannot be the final candidate for activation. It is worth mentioning that the idea behind this elimination is similar to the LsKO algorithm, but we apply the principle on candidate neurons, not active neurons, and so the elimination rule becomes different. We summarize the recall method in algorithm 1; the complete version is in Mofrad et al. (2016).

Some important parameters of a memory are diversity, capacity, and efficiency. Based on Gripon and Berrou (2011), diversity is the number of learned patterns, capacity is the maximum amount of data learned in bits,

and efficiency is the ratio between capacity and the amount of information used by the network when $M = M_{\max}$ ($M_{\max}$ is an upper bound for the number of learned patterns). Since the maximum number of edges in the clique-based model is $Q = \frac{(c-1)n^2}{2c}$ and the edges are binary, this amount of available memory ideally allows the storage of

$$M_{\max} = \frac{(c-1)n^2}{2ck} = \frac{(c-1)n^2}{2c^2 \log_2(\frac{n}{c})} \tag{2.1}$$

binary patterns, where $k = c\kappa = c\log_2(l)$, and $l = \frac{n}{c}$.

3 Extension of a Clique-Based Network to a Nested-Clique Network

Suppose the network of n neurons splits into $c_1 \times c_2$ clusters, each of size $l = \frac{n}{c_1 \times c_2}$, where l is set to be a power of 2 and $\kappa = \log_2(l)$. These clusters split into c_1 superclusters of size c_2. Each binary pattern m of length $k = c_1 \times c_2 \times \kappa$ maps to

$$\begin{aligned} m &= (m_{11}m_{12}\cdots m_{1c_2}; m_{21}m_{22}\cdots m_{2c_2}; \cdots m_{c_11}m_{c_12}\cdots m_{c_1c_2}) \\ &\longrightarrow (f(m_{11})f(m_{12})\cdots f(m_{1c_2}); f(m_{21})f(m_{22})\cdots f(m_{2c_2}); \\ &\quad \cdots f(m_{c_11})f(m_{c_12})\cdots f(m_{c_1c_2})), \end{aligned}$$

where f is again a map from subpatterns of length κ to a unique neuron

$$f : \{0,1\}^\kappa \to \mathcal{A}.$$

The learning process for pattern m is then to establish edges between active neurons in each of the c_2 clusters in c_1 superclusters; the edges will be

$$(f(m_{is}), f(m_{it})), \text{ for } 1 \le i \le c_1 \text{ and } 1 \le s \ne t \le c_2.$$

We call these edges "short connections." Similarly, active neurons in equivalent clusters of different superclusters connected with edges will be called "long connections":

$$(f(m_{is}), f(m_{js})), \text{ for } 1 \le i \ne j \le c_1 \text{ and } 1 \le s \le c_2.$$

This makes a c_1-cliques-of-c_2-cliques or $K_{c_1}[K_{c_2}]$ (see Figure 2).

Different values of n, l, c_1, and c_2 affect the memory retrieval performance, but there is no difference between a network with $c_1 = a, c_2 = b$ and learning patterns as a-cliques-of-b-cliques and a network with $c_1 = b$, $c_2 = a$ and

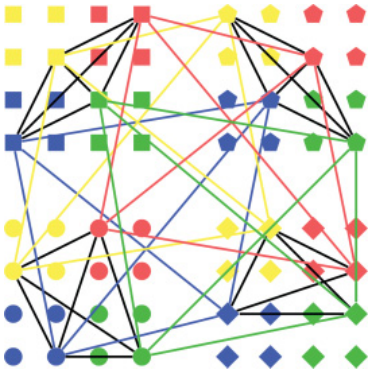


Figure 2: The learning process in a network with 64 neurons split into 4 superclusters, each with 4 clusters of 4 neurons. The 4-cliques-of-4-cliques reflect the pattern $m = (11, 01, 10, 00; 00, 10, 01, 11; 10, 00, 11, 11; 01, 11, 00, 01)$. In each supercluster, a clique of size 4 (in black) is established by short connections; activated neurons in the equivalent clusters (same colors) are connected with long connections and form a clique of size 4. Altogether the patterns are represented as a 4-cliques-of-4-cliques or $K_4[K_4]$.

learning patterns as b-cliques-of-a-cliques. It can easily be verified that after the learning process, both topologies are the same, and one can rearrange the clusters to have c_2 superclusters, each with c_1 clusters. It is equivalent to the learning pattern

$$m' = (m_{11}m_{21}\cdots m_{c_1 1}; m_{12}m_{22}\cdots m_{c_1 2}; \cdots m_{1c_2}m_{2c_2}\cdots m_{c_1 c_2})$$

instead of m, which produces exactly the same connections, but short connections become long connections and vice versa. From the information-theoretic view, there is no difference between learning a set of patterns like m' instead of the original set of patterns. However, since in general, short connections are preferable to long connections, we choose $c_2 \geq c_1$ hereafter.

Retrieval is basically the same as the clique-based version. The only difference is in the neuron removal from candidate sets, which is pointed out as ***nested** in algorithm 1. The **if** condition would be changed to

---

***nested** **if** n_{ij} *does not connect to all other remained candidate sets in the same supercluster and all other remained candidate sets in the equivalent clusters in other superclusters with at least one edge* **then**

n_{ij} is kicked out from $T(i)$.

We can use the same argument for memory parameters when the network is a nested clique. The maximum number of possible edges is $Q' = \frac{n^2(c_1+c_2-2)}{2c_1 \times c_2}$, which gives an upper bound for diversity,

$$M'_{\max} = \frac{n^2(c_1+c_2-2)}{2(c_1 \times c_2)k} = \frac{n^2(c_1+c_2-2)}{2(c_1 \times c_2)^2 \log_2(\frac{n}{c_1 \times c_2})}, \tag{3.1}$$

where $k = c_1 \times c_2 \times \kappa = c_1 \times c_2 \log_2(l)$, and $l = \frac{n}{c_1 \times c_2}$.

Note that the clique-based model can be considered a special case of the nested-clique model where $c_1 = 1$ and $c_2 = c$. It can be easily seen that in this case, $Q' = Q$ and $M'_{\max} = M_{\max}$.

To compare the current work with local coding and precoding methods, which we applied in previous work (Mofrad et al., 2016), we need a brief review of those. Consider that a set of patterns of length $c\kappa$ is learned to a GB network with $c \times 2^\kappa$ neurons; c clusters of size $l = 2^\kappa$. Local coding converts each subpattern of length κ to a code word of size κ' from a chosen code with appropriate Hamming distance. The only difference with an uncoded GB is to project a code word to a neuron instead of a random subpattern. The learning process produces the same edge set W, but in retrieval, the distance between code words enhances the performance at the local level (i.e., within clusters). By the precoding technique, a set of patterns, each of length $c\kappa$, maps to a set of code words of a chosen code where code word length $\mathcal{N} > c\kappa$. Therefore, if we choose a proper code such that $\frac{\mathcal{N}}{c}$ is a natural number, then we can learn code words in a GB network of $c \times 2^{(\mathcal{N}/c)}$ neurons. As can be seen in precoding, n and l are not preserved like local coding, but for the same number of clusters, the number of edges (or Q), remains equal.

Precoding enhances local retrieval compared to the uncoded version of GB, but it is weaker than the local coding model. The main performance gain of precoding comes from generating a higher distance between cliques associated with different patterns, so precoding is suitable when a higher density is required. Both local coding and precoding manipulate the data to be learned not the network topology.

The nested-clique model is an extension of the GB model, and as we mentioned, GB can be considered a special case of the nested-clique model. We can look at the nested-clique technique in three ways. First, consider c_2 parallel GB memories with $c = c_1$ and $l = 2^\kappa$ such that c_2 individual sets of random patterns of size d are learned to them. If we consider them as superclusters and add edges to equivalent clusters (long edges), we achieve a nested-clique memory for which new edges act as the second source for association, and retrieval is enhanced in comparison to the parallel scenario.

So although we have longer patterns, the new patterns can be seen as a combination of smaller (meaningful) patterns.[4]

As the second angle, we can consider a GB network with c clusters each of size l and a given c_2, where $\frac{l}{c_2}$ is a power of 2. To achieve a nested-clique version, instead of the mapping $f : \{0, 1\}^{\kappa} \to \mathcal{A}$ where $\kappa = \log_2(l)$, a new map can be used that activates c_2 neurons in each cluster instead of just one: $f' : \{0, 1\}^{\kappa'} \to \mathcal{A}'$ where $\kappa' = \log_2(\frac{l}{c_2})$ and $|A'| = \frac{l}{c_2}$. So by adding edges between active neurons in each cluster (short connections) and connecting active neurons of equivalent clusters, the association within each cluster is higher, and therefore the retrieval rate will be enhanced.

Note that in both arguments, we add some new connections and indeed involve each subpattern in two cliques, (two constraints in coding theory terminology), which leads to higher performance. The idea of using two encodings of the same message in order to gain some benefit is not new in coding theory (see Turbo codes, for instance, in Berrou & Glavieux, 1996, and for neural networks, see Jiang et al., 2016).

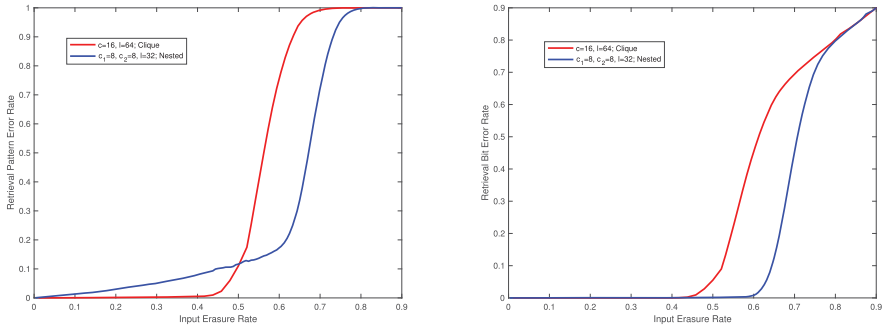
In the above arguments, we added edges to the GB network to make a nested-clique with better performance; as another possibility, one may degenerate a clique-based network to achieve a nested-clique network. Consider a GB network with $c_1 \times c_2$ clusters, so each pattern is projected to a $(c_1 \times c_2)$-clique with $\frac{(c_1 \times c_2)(c_1 \times c_2 - 1)}{2}$ connections. If we partition cliques into c_1 superclusters of c_2 cliques and take off the connections between nonequivalent clusters, we will achieve a nested-clique structure. This time, since we lose connections and therefore information, the recall performance would be weaker. Degenerated cliques are addressed in Jiang et al. (2016) where it was shown that (in Jiang's Figure 3) the retrieval performance in the presence of partial erasure decreases for degenerated cliques as expected.

From the three different ways to achieve a nested-clique network from a GB network, it can be seen that the number of connections plays a key role. To make a fair comparison of the two structures, we fixed the amount of available memory, or the number of edges, and compared nested-clique and clique-based versions in the simulation part.

4 Simulation Results

To see the retrieval performance of the proposed associative memory and compare clique-based and nested-clique-based scenarios, we fixed the number of connections or the available memory Q for both scenarios. The

[4] For example, each cluster may represent a "word," and each supercluster represents a meaningful sentence, which is a combination of c_1 words. If we have c_2 such memories and the long connections represent similar word classes (all are nouns or verbs, for example), then we have an extra knowledge, which assists in recalling the sentences in superclusters.



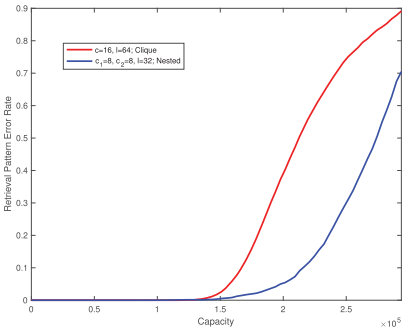
(a) Pattern Error rate (b) Bit error rate

Figure 3: Number of clusters for the red curve is $c = 16$. Each subpattern is of length $\kappa = 6$, so there are 64 neurons per cluster with pattern length 96. For the blue curve, the parameters are $c_1 = 8$, $c_2 = 8$, and $\kappa = 5$, so the number of neurons per cluster is 32 and the pattern length is 320. See Table 1 for a comparison of parameters.

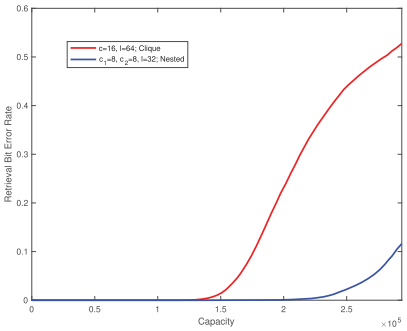
symbols in the patterns are considered i.i.d. random variables. Because the data set is chosen randomly, we repeat the experiment 2500 times and compute the average to have more reliable results (i.e., we randomly choose 2500 patterns from the learning set as the input and partially erase them). We then tried retrieving the chosen pattern. If the pattern is completely retrieved, the algorithm is successful; otherwise, it fails. Both pattern error rate, which is the rate of unsuccessful retrieval, and bit error rate, which refers to the probability of one bit being erased after retrieval, are evaluated.

The first comparison is done for similar amounts of learned data (in bits) for different erasure rates, (see Figure 3), and then a comparison is made based on different amounts of learned data when the erasure rate is fixed to 0.6 (see Figure 4). The network parameters are provided in Table 1 for Figure 3 and in Table 2 for Figure 4.

In Figure 3, the number of possible edges for the GB model with $c = 16$ and $l = 64$ is 4.9×10^5. For the nested-clique structure with $c_1 = 8, c_2 = 8$, and $l = 32$, we have 4.6×10^5 possible edges, and the memory used (Q) for both is approximately the same. Since the length of patterns is longer (about three times) in nested-clique models, we choose higher diversity (about 3 times) for the GB model to have the same capacity and efficiency to compare the retrieval performance in different erasure rates. In Table 1, parameters are compared when the erasure rate is 0.6. With these parameters, we see that the retrieval pattern error rate in a nested clique is 6 times less than the GB model, and for the bit error rate, this value becomes even better—53 times less. Note that the bit error rate in the worse case equals the erasure rate while the pattern error rate in the worst case equals 1.



(a) Pattern error rate in retrieval



(b) Bit error rate in retrieval

Figure 4: The number of clusters for the red curve is $c = 16$. Each subpattern is of length $\kappa = 6$, so there are 64 neurons per cluster with pattern length 96. For the blue curve, the parameters are $c_1 = 8$, $c_2 = 8$, and $\kappa = 5$, so the number of neurons per cluster is 32 and the pattern length is 320. The erasure probability is fixed at 0.6, and retrieval for different capacity is depicted. See Table 2 for a comparison of parameters.

Table 1: Comparison of Performance between the GB Model and the Nested Clique for the Same Capacity.

Model	GB	Nested Clique	Ratio
Memory used (Q)	4.9×10^5	4.6×10^5	≈ 1
Neurons (n)	1024	2048	$\times 2$
Pattern length	96	320	$\times 3$
Pattern error rate	0.79	0.13	$\div 6$
Bit error rate	0.48	0.009	$\div 53$
Diversity	2600	730	$\div 3.5$
Capacity	249,600	233,600	≈ 1
Efficiency	0.51	0.51	≈ 1

Note: In the GB model $c = 16$ and $\kappa = 6$; in the nested-clique model $c_1 = 8$, $c_2 = 8$, and $\kappa = 5$ where the erasure rate is fixed to 0.6.

In Figure 4 for the same networks and memory used (Q), we fixed the erasure rate to 0.6 and compare the two networks by the capacity factor, so diversity, capacity, and efficiency are not the same here. In Table 2 we fixed the pattern error rate to see to what extent networks can learn and recall with a pattern error rate less than 0.1. The GB model has a higher diversity 2.6 times more, but since the pattern length is smaller, the capacity in the nested clique is still better (1.3 times more than GB). As a consequence, the efficiency of the nested clique outperforms GB. So if the concern is the total information that the network is able to memorize and then retrieve in

Table 2: Comparison of Performance between the GB Model and the Nested Clique for the Same Capacity.

Model	GB	Nested Clique	Ratio
Memory used (Q)	4.9×10^5	4.6×10^5	≈ 1
Neurons (n)	1024	2048	$\times 2$
Pattern length	96	320	$\times 3$
Pattern error rate	0.102	0.1	1
Bit error rate	.06	0.002	$\div 30$
Diversity	1740	666	$\div 2.6$
Capacity	167,040	213,120	$\times 1.3$
Efficiency	0.34	0.46	$\times 1.35$

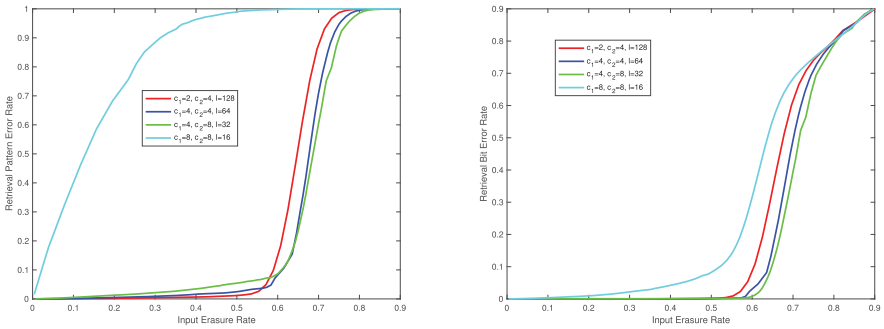
Note: In the GB model, $c = 16$ and $\kappa = 6$. In the nested-clique model, $c_1 = 8$, $c_2 = 8$, and $\kappa = 5$, where the erasure rate is fixed to 0.6 and the pattern error rate equals 0.1, which is acceptable.

the presence of partial erasure, the nested clique is better, but if diversity is the most important parameter, since GB memorizes smaller patterns, it can learn and recall more patterns within a fixed amount of memory used (Q).

Moreover, a comparison between different nested-clique scenarios for a fixed number of neurons was considered. In Figures 5 and 6 for $n = 1024$, four nested-clique configurations are compared. For the same capacity (80,640 bits) in Figure 5, we see that the nested-clique model with $c_1 = 8, c_2 = 8$, and $l = 16$ does not have good results compared to the other three configurations. The reason might be the very small size of clusters. Depending on which parameter is more favorable, one might choose a different configuration. As we can see in Table 3, the $c_1 = 2, c_2 = 4$ model has better diversity than $c_1 = 4, c_2 = 4$, which is better than $c_1 = 4, c_2 = 8$. However, the pattern retrieval rate and efficiency is in the reverse order. Since we care about capacity and the retrieval rate, the best configuration in this simulation will be $c_1 = 4, c_2 = 8$.

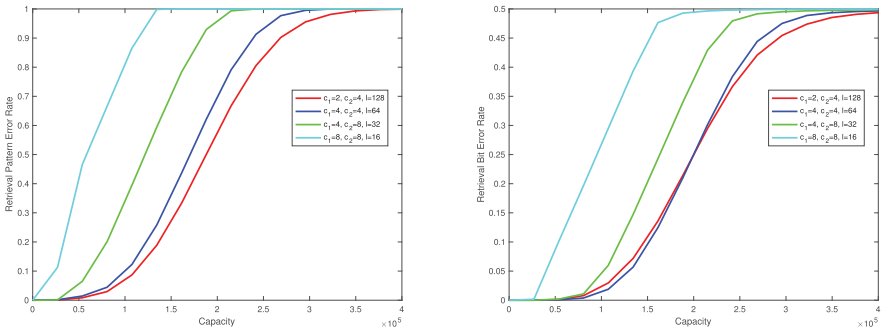
For the fixed erasure rate 0.5 in Figure 6, we see that the larger cluster size results in a better retrieval error. We fixed the acceptable retrieval error rate to 0.05 in Table 4.

The $c_1 = 2, c_2 = 4$ model has better diversity than $c_1 = 4, c_2 = 4$, but both have the same capacity. Since the number of possible connections in $c_1 = 4, c_2 = 4$ is fewer, the efficiency is better. So for the same amount of information, one can choose between better diversity or better efficiency by choosing $c_1 = 2, c_2 = 4$ or $c_1 = 4, c_2 = 4$. $c_1 = 4, c_2 = 8$ has good efficiency compared to the $c_1 = 2, c_2 = 4$ and $c_1 = 4, c_2 = 4$; however, it has less diversity and capacity. This scenario is able to learn longer patterns instead. The last scenario, $c_1 = 8, c_2 = 8$, can learn long patterns; however, its diversity, capacity, and efficiency are not comparable to the other configurations. One



(a) Pattern error rate in retrieval (b) Bit error rate in retrieval

Figure 5: A comparison of different nested-clique scenarios by changing c_1 and c_2 while the number of neurons are fixed.



(a) Pattern error rate in retrieval (b) Bit error rate in retrieval

Figure 6: A comparison of different nested-clique scenarios with a fixed number of neurons with the erasure probability set to 0.5.

might choose based on which parameter is more important; however, the $c_1 = 2, c_2 = 4$ configuration shows better capacity and diversity under the conditions of an erasure rate of 0.5 and a retrieval pattern rate of 0.05.

5 Conclusion

The nested-clique neural associative memory introduced in this letter is an extension of the clique-based (GB) mode. We have compared the retrieval capability of the two networks in the presence of partial erasures. It is possible to achieve a nested-clique network from a GB (or a set of GB) network(s) by adding extra connections, which makes the network stronger and more robust against erasure. It is also possible to remove some connections and

Table 3: Comparison between Different Nested-Clique Scenarios for Fixed Number of Neurons and Capacity When the Erasure Rate Is Fixed to 0.6.

Model	$c_1 = 2, c_2 = 4$	$c_1 = 4, c_2 = 4$	$c_1 = 4, c_2 = 8$	$c_1 = 8, c_2 = 8$
Memory used (Q)	2.6×10^5	2×10^5	1.6×10^5	1.1×10^5
Neurons (n)	1024	1024	1024	1024
Pattern length	56	96	160	256
Pattern error rate	0.129	0.0748	0.0876	0.998
Bit error rate	0.0706	0.024	0.0063	0.362
Diversity	1440	840	504	315
Capacity	80,640	80,640	80,640	80,640
Efficiency	0.31	0.41	0.5	0.73

Table 4: A Comparison of Different Nested-Clique Scenarios with a Fixed Number of Neurons Where the Erasure Rate Is Fixed to 0.5 and the Accepted Retrieval Pattern Error Rate Is Set to 0.05.

Model	$c_1 = 2, c_2 = 4$	$c_1 = 4, c_2 = 4$	$c_1 = 4, c_2 = 8$	$c_1 = 8, c_2 = 8$
Memory used (Q)	2.6×10^5	2×10^5	1.6×10^5	1.1×10^5
Neurons (n)	1024	1024	1024	1024
Pattern length	56	96	160	256
Pattern error rate	0.046	0.05	0.05	0.004
Bit error rate	0.006	0.0025	0.0012	0.0
Diversity	1920	1120	504	105
Capacity	107,520	107,520	80,640	26,880
Efficiency	0.41	0.55	0.504	0.244

degenerate a clique-based network to achieve a nested-clique network; in such a case, the nested-clique network is less robust against erasure. For a fair comparison in simulation, we fixed the number of connections and capacity and saw that the nested-clique structure outperforms the clique-based model. The nested-clique model in this case is able to learn longer patterns while its diversity (due to fixed capacity) is smaller. Both local coding and precoding techniques that improve the clique-based model can be used in the nested-clique model in the same manner if one needs to store more information and protect it from strong erasures. Local coding is especially helpful in the case that the erasure rate and diversity are both high. It is worth mentioning that one may consider other types of nested graphs—for instance, $K_{c_1}[C_{c_2}]$, which has a cycle in the lower layer. Moreover, for larger networks one expects to extend the nested-clique structure to more layers (e.g., $K_{c_1}[K_{c_2}[K_{c_3}]]$).

References

Aliabadi, B. K., Berrou, C., Gripon, V., & Jiang, X. (2014). Storing sparse messages in networks of neural cliques. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(5), 980–989.

Berrou, C., & Glavieux, A. (1996). Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications*, *44*(10), 1261–1271.

Bohland, J. W., & Minai, A. A. (2001). Efficient associative memory using small-world architecture. *Neurocomputing*, *38*, 489–496.

Danielsen, L. E., & Parker, M. G. (2004). Spectral orbits and peak-to-average power ratio of Boolean functions with respect to the {I, H, N} n transform. In *Proceedings of the International Conference on Sequences and Their Applications* (pp. 373–388). New York: Springer.

Gardner, E. (1989). Optimal basins of attraction in randomly sparse neural network models. *Journal of Physics A: Mathematical and General*, *22*(12), 1969–1974.

Gripon, V., & Berrou, C. (2011). Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, *22*(7), 1087–1096.

Gripon, V., Heusel, J., Löwe, M., & Vermet, F. (2015). *A comparative study of sparse associative memories*. arXiv:1512.08892.

Hilgetag, C. C., & Goulas, A. (2016). Is the brain really a small-world network? *Brain Structure and Function*, *221*(4), 2361–2366.

Hopfield, J. J. (2008). Searching for memories, sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. *Neural Computation*, *20*(5), 1119–1164.

Jiang, X. (2014). *Storing sequences in binary neural networks with high efficiency*. PhD diss., Université de Bretagne Occidentale.

Jiang, X., Gripon, V., Berrou, C., & Rabbat, M. (2016). Storing sequences in binary tournament-based neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(5), 913–925.

Mofrad, A. A., Parker, M. G., Ferdosi, Z., & Tadayon, M. H. (2016). Clique based neural associative memories with local coding and pre-coding. *Neural Computation*, *28*, 1–21.

Stauffer, D., Aharony, A., da Fontoura Costa, L., & Adler, J. (2003). Efficient Hopfield pattern recognition on a scale-free neural network. *European Physical Journal B-Condensed Matter and Complex Systems*, *32*(3), 395–399.

Received July 7, 2016; accepted January 30, 2017.

Paper III

7.3 On Neural Associative Memory Structures: Storage and Retrieval of Sequences in a Chain of Tournaments

Asieh Abolpour Mofrad, Samaneh Abolpour Mofrad, Anis Yazidi, and Matthew G. Parker

Mofrad, A. A., Mofrad, S. A., Yazidi, A., & Parker, M. G. On Neural Associative Memory Structures: Storage and Retrieval of Sequences in a Chain of Tournaments. Accepted to be published in Neural Computation.

On Neural Associative Memory Structures: Storage and Retrieval of Sequences in a Chain of Tournaments

Asieh Abolpour Mofrad[1, *]
Samaneh Abolpour Mofrad[2, 3, *]
Anis Yazidi[4, 5]
Matthew Geoffrey Parker[1]

[1] The Selmer Center, Dept. of Informatics, University of Bergen, Bergen, Norway.
[2] Dept. of Computer Science, Electrical Engineering, and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway.
[3] Mohn Medical Imaging and Visualization Center, Haukeland University Hospital, Bergen, Norway.
[4] Dept. of Computer Science, OsloMet - Oslo Metropolitan University, Oslo, Norway.
[5] Dept. of Plastic and Reconstructive Surgery, Oslo University Hospital, Oslo, Norway.
[*] These authors contributed equally.

Keywords: Auto-associative memories, clique-based neural networks, tournament-based neural networks, sequence storage

Abstract

Associative memories enjoy many interesting properties in terms of error correction capabilities, robustness to noise, storage capacity and retrieval performance and their usage spans over a large set of applications. In this article, we investigate and extend Tournament-Based Neural Networks, originally proposed by Jiang et al. (2016), which is a novel sequence storage associative memory architecture with high memory efficiency and accurate sequence retrieval. We propose a more general method for learning the sequences which we call Feedback Tournament-Based Neural Networks. The retrieval process is also extended to both directions: forward and backward, i.e. any large-enough segment of a sequence can produce the whole sequence. Furthermore, two retrieval algorithms, Cache-Winner and Explore-Winner are introduced to increase the retrieval performance. Through simulation results, we shed light on the strengths and weaknesses of each algorithm.

1 Introduction

Neural associative memory is a type of neural networks which is capable of memorizing (learning) a set of patterns and retrieving them from their corresponding noisy or incomplete versions. The term *association* refers to the linkage of two or more pieces of information. Hopfield neural network (Hopfield, 1982) was among the first designed artificial neural network with auto-associative memories which is able to retrieve information given only some partial clues as well as reconstruct perturbed patterns. Hopfield neural networks have some drawbacks such as being biologically implausible, due to the fully connected structure, low efficiency and spurious memories (see, e.g., Hoffmann, 2019, and references therein). To improve Hopfield network many variants of it have been proposed in the literature (see, e.g. Maurer et al., 2005; Berrou & Gripon, 2010; Krotov & Hopfield, 2016; Kim et al., 2017). Due to the *sparse coding* in the brain (for sparse coding see, e.g, Olshausen & Field, 2004; Rinkus, 2010), sparse associative memories are considered more biologically plausible models (Gripon et al., 2016; Hoffmann, 2019).

Gripon & Berrou (2011) proposed novel sparse neuro-inspired associative memories that organize neurons into clusters and memorize patterns using the concept of cliques (see also, Hopfield, 2008, for another clique-based network model of associative memory). This model, also referred to as GB model or Clustered Cliques Networks (CCNs), has fundament in information theory (Gripon & Berrou, 2012) and bears similarity to the Willshaw-type model (Willshaw et al., 1969) where sparse patterns and binary connections are considered. These models have been further developed in the literature (e.g. Aliabadi et al., 2014; Boguslawski et al., 2014; Jarollahi et al., 2014, 2015; Jiang et al., 2015, 2016; Mofrad et al., 2015, 2016; Mofrad & Parker, 2017; Berrou & Kim-Dufor, 2018), and used in many applications, such as solving feature correspondence problems (Aboudib et al., 2016), devising low-power content-addressable memory (Jarollahi et al., 2015), oriented edge detection in image (Danilo et al., 2015), image classification with Convolutional Neural Networks (Hacene et al., 2019), finding all matches of a probe in a database (Hacene et al., 2017), to mention a few. Furthermore, they were implemented on a general purpose graphical processing unit (GPU) (Yao et al., 2014), in 65-nm CMOS (Larras et al., 2018), and in distributed smart sensors architectures (Larras & Frappé, 2020). Therefore, CCN models can be referred to as an important brain-inspired memory system (Berrou et al., 2014) that became a basis for a wide range of research in associative memory models.

Learning and retrieval of temporal sequences in neural networks is a fundamental property of human intelligence which is studied through different approaches (see, e.g., Brea et al., 2011; Hawkins et al., 2009; Maurer et al., 2005; Jiang et al., 2016). Tournament-based Neural Network (TNN) (Jiang et al., 2016) is an extension of the clique-based approach to associative memories which have oriented connections, and therefore the ability to store sequential information (see also, Marques et al., 2017, for an implementation on the GPU). The novel structure of TNN is not only a sequence storage with high memory efficiency, but also a more compatible model with the neuronal signal propagation in the brain via oriented connections (see also Hawkins et al., 2009; Hawkins & Ahmad, 2016, for biologically plausible memory sequence structures).

In this paper, we improve the TNN architecture by proposing a more general struc-

ture, named Feedback TNN, as well as more accurate retrieval algorithms. The original TNN can be considered as a special case of Feedback TNN, with zero feedback connections. For retrieval, obviously, a less number of random selections during retrieval results into less component and sequence error at the end. The Cache-Winner retrieval revisits and changes some previous randomly selected components, in case an error is detected during retrieval. On the other hand, Explore-Winner reduces the randomness in decisions by considering the consequences of each decision. The idea behind the Cache-Winner technique can be illustrated in simple terms by drawing analogy with human decision making: imagine a person who makes a decision fast and then, if he realizes a mistake, tries to resolve it by manipulation of past decisions. On the other hand, Explore-Winner has the analogy with a rather careful decision-maker who investigates the consequences of all possible decisions at the time and then makes the best possible decision. In terms of achieving accurate sequence retrieval, both proposed retrieval techniques are superior to the Winner, which literally makes a random decision in the case of equal chance situations, and continues without further actions even when realizing a mistake later.

It is also known that the brain is able to follow the previously stored sequences, from any given point forward, and somewhat, also backwards (see, e.g. Hawkins & Blakeslee, 2007). The other contribution of this paper is introducing Feedback-Backward retrieval method which makes our model more biologically plausible. Using Feedback-Backward retrieval, the model gains the capability of retrieval of the whole sequence, given a sub-sequence, no matter its location. The Feedback-Backward retrieval is more compatible with the Feedback TNN, but works well with the original TNN as shown in the results. Backward retrieval, therefore, adds more capabilities to these types of sequence storage structures, and makes them more similar to brain functioning.

The paper is organized as follows: in section 2 we briefly survey the CCN and TNN structures. In section 3, different learning and retrieval algorithms are explained. The simulation results are provided in section 4, and afterwards, in section 5, discussion and concluding remarks are presented.

2 Background

In this section, first the clustered clique-based neural network structure is described in section 2.1. These types of networks are able to store and retrieve the fixed length patterns. Next, in section 2.2, tournament-based neural networks which have the ability to store and retrieve sequences is surveyed.

2.1 Clustered Clique Networks (CCNs)

In Clustered Clique Networks (CCNs) the way the neurons are organized within clusters, and the sparsity of the encoding used for storing patterns in cliques, result into large storage diversity, i.e. number of storable patterns, high capacity, i.e. the amount of storable information, and strong robustness against erasures and errors (Gripon & Berrou, 2011; Jarollahi et al., 2015; Gripon et al., 2016).

Formally, the structure of CCNs consists of n neurons divided into c clusters with possibility of different sizes. The input patterns are formed from a pre-defined alphabet $\mathcal{A}$ where the number of neurons in each cluster matches the size of used alphabet $|\mathcal{A}|$. For simplicity all clusters are considered to have the same number of neurons, say $l = n/c$, and therefore the same alphabet size $|\mathcal{A}| = l$. The j^{th} neuron in the i^{th} cluster is denoted by n_{ij} and it has an associated value, $v(n_{ij})$, equals one if it is activated, and zero otherwise; where $1 \leq i \leq c$ and $1 \leq j \leq l$. Let $\mathcal{P}$ be the set of patterns to be stored where pattern $p \in \mathcal{P}$ contains c sub-patterns, i.e. $p = p_1 p_2 \cdots p_c$; for $p_i \in \mathcal{A}$.

The learning process starts by assigning a unique set of neurons -one per cluster- to each $p \in \mathcal{P}$:

$$p = p_1 p_2 \cdots p_c \rightarrow (f(p_1), f(p_2), \cdots, f(p_c))$$
$$\text{where} \quad f : \{p_i\} \rightarrow \{n_{ij} | 1 \leq j \leq l\}.$$

Learning proceeds by activation of the selected neurons, i.e. $v(n_{ij}) = 1$, and forming a clique by connecting the selected c active neurons to each other through binary edges. As a result, the learning process generates a set of binary edges

$$\mathcal{W} = \{\omega_{(ij)(i'j')} | \text{ if } i \neq i' \text{ and } \exists\, p \in \mathcal{P} \text{ s.t. } f(p_i) = n_{ij} \text{ and } f(p_{i'}) = n_{i'j'}\},$$

where $\omega_{(ij)(i'j')}$ is an edge between n_{ij} and $n_{i'j'}$.
The edge $\omega_{(ij)(i'j')}$ belongs to $\mathcal{W}$ independently from the number of patterns that use both n_{ij} and $n_{i'j'}$ neurons, but only if there exists such a pattern. Figure 1 illustrates the storing process in clique-based networks.

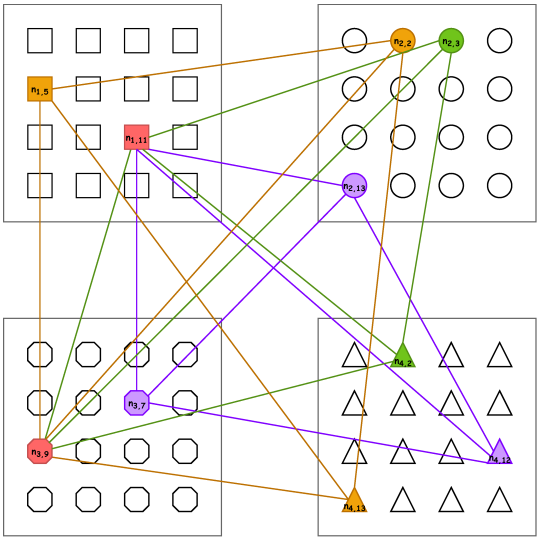


Figure 1: The learning process of three patterns, in a network with $c = 4$ clusters and $l = 16$ neurons per cluster. Node $n_{i,j}$ refers to the j^{th} neuron in the i^{th} cluster. Each clique represents one of the three $(4, 1, 8, 12)$, $(10, 2, 8, 1)$, and $(10, 12, 6, 11)$ patterns with yellow, green, and purple respectively. Coloured nodes refer to the activation of neurons for at least one pattern. The red nodes, $n_{1,11}$ and $n_{3,9}$, belong to two patterns. Note that it is not possible to retrieve the patterns by finding a unique clique using only one of these red nodes.

The recall or retrieval phase of a possibly distorted version of a learnt pattern, $\hat{p}$, is based on finding the closest match from $\mathcal{P}$. Depending on the type of distortion, various retrieval methods might be used (see, Aboudib et al., 2014), however, in general the recall procedure consists of local and global phases. The local phase aims to find the most probable neurons in different clusters, using information from $\hat{p}$ or incoming connections from previously activated neurons, and activate them, i.e. $v(n_{ij}) = 1$. The global phase is to recall the established edges in $\mathcal{W}$ that have an end in activated neurons. This procedure alternate between global and local retrieval to gradually complete the clique and therefore the pattern.

It is noteworthy that other sparse structures were presented by Aliabadi et al. (2014), according to which, $c \ll \chi$ where $\chi = n/l$ denotes the number of clusters and c was used to denote a smaller set of clusters for which a sparse pattern is mapped into. Retrieval, in this case, would be more complicated and various scenarios could be considered (see, e.g., Aboudib et al., 2014; Jiang, 2014). For instance, the winner-take-all rule activates neurons with the highest activity (or maximum score), whilst Losers Kicked-Out rule (LsKO) eliminates active neurons with less activity using a threshold filter (see Jiang, 2014, for details).

2.2 Tournament-Based Neural Network (TNN)

An extension of the CCNs (Jiang et al., 2016) is proposed by using directed edges between clusters in such a way that the network can store sequential information in a tournament-based[1] neural network. In a chain of tournaments of order c and degree r, denoted by $\mathcal{T}_r(c)$, each node is directed clockwise to its r consecutive neighbors; see Figure 2 with $c = 8$ and $r = 3$ for a sample chain of tournaments. A TNN can then be seen as a concatenation of tournaments of size $r + 1$.

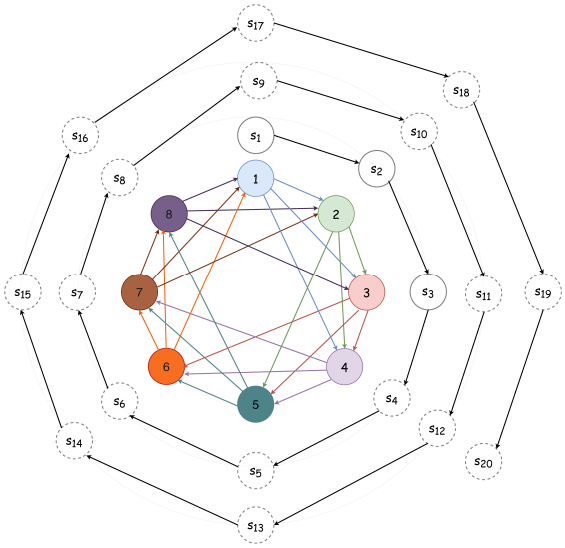


Figure 2: An illustration of a chain of tournaments, $\mathcal{T}_3(8)$, for storing sequences of length 20. The eight clusters are represented by colored circles, and each arrow represents a set of possible connections between nodes within the clusters. The clusters construct eight tournaments of size $r + 1 = 4$. For instance, clusters that have been shown with $1, 2, 3, 4$ make one tournament starting from cluster 1, and clusters labeled with $7, 8, 1, 2$ involve in another tournament starting from cluster 7. A sequence of length 20 and the assigned clusters for each component s_i are represented around the network. Given the first r components (s_1, s_2, s_3) with solid circles, the retrieval algorithm could retrieve the rest sequentially using the tournament connections. This figure is based on (Jiang et al., 2016, Fig. 5).

In order to store a set of sequences, $\mathcal{S}$, in a chain of tournaments, we suppose that each sequence $s \in \mathcal{S}$ contains L component, i.e. $s = s_1 s_2 \cdots s_L$; for $s_t \in \mathcal{A}$, $t = 1, 2, \ldots, L$, and $|\mathcal{A}| = l$.

By labeling clusters from 1 to c, the learning process could be explained as follows. First a unique sequence of neurons must be assigned to each $s \in \mathcal{S}$ by using function

[1] In graph theory, by assigning direction to all edges of a complete graph, a tournament can be achieved.

$f = (f_1, \cdots, f_c)$, where $f_i,\ i = (t-1 \bmod c) + 1$, maps a component s_t, to a unique neuron n_{ij} in cluster i:

$$f_i : \{s_t\} \rightarrow \{n_{ij} | 1 \leq j \leq l, \},\ 1 \leq i \leq c,$$

therefore,

$$f(s) = (f_1(s_1), f_2(s_2), \cdots, f_c(s_c), \cdots, f_{(L-1 \bmod c)+1}(s_L))$$

Learning continues by connecting neuron n_{ij} to neuron $n_{i'j'}$ at passage π as follows

$$n_{ij} \rightarrow n_{i'j'},\ \text{if:} \begin{cases} f_i(s_{(i+(\pi-1)c)}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{and, } 1 \leq \delta_i(i') \leq r \tag{1}$$

where $\delta_i(i') = (i' - i) \bmod c$, and $1 \leq \pi \leq \lfloor \frac{L}{c} \rfloor$.

In general, for $s \in \mathcal{S}$, if the above conditions are satisfied for a given π such that $n_{ij} \rightarrow n_{i'j'}$, we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 1$, which means that n_{ij} is connected to $n_{i'j'}$, in sequence s, otherwise we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 0$. In Figure 2, s_2 is connected to s_3 in passage $\pi = 1$, but not to s_{11} (in passage $\pi = 2$), and s_{19} (in passage $\pi = 3$) in the same sequence s, for instance. So the neighboring connections are defined based on both s and π values.

At the end of learning or storing process, the network has the following connections:

$$\mathcal{W} = \{\omega_{(ij)(i'j')} | \text{ if } \exists\ s \in \mathcal{S},\ \text{and } \exists\ \pi \in [1 : \lfloor \frac{L}{c} \rfloor] \text{ s.t. } N_{s,\pi}(n_{ij}, n_{i'j'}) = 1\} \tag{2}$$

where $\omega_{(ij)(i'j')}$ is a directed edge from n_{ij} to $n_{i'j'}$ and $1 \leq i, i' \leq c,\ 1 \leq j, j' \leq l$ (see Algorithm 1 for the learning process).

A stored sequence retrieval process could start with any subsequence of r consecutive components and the activation of a component in the following cluster relies on the connections of r previous clusters. If the given subsequence is not the first r components of the sequence, the retrieval algorithm requires the information of the location of clusters. In Figure 2, the first three components s_1, s_2, and s_3 are shown with solid circles, and the components to be retrieved are shown with dashed circles.

The proposed retrieval procedure is sequential using a Winner-Takes-All (WTA) decision at each step. For brevity, we call this retrieval *Winner* in the rest of paper (see Algorithm 2).

3 Structures and Algorithms

The original learning and retrieval algorithms for TNN that were proposed by (Jiang et al., 2016) are reported in section 3.1. In sections 3.1.1 and 3.1.2, the newly proposed retrieval algorithms Winner-Cache and Winner-Explore are provided respectively. Feedback TNN structure along with its corresponding learning and retrieval algorithms, Feedback-Forward and Feedback-Backward, are presented in section 3.2. Finally, the

error types that are used for evaluation of structures are addressed at the end of this section (section 3.3).

3.1 Learning and Retrieval Algorithms in TNN

TNN structure, which is explained in section 2.2, is summarized by Algorithm 1 and Algorithm 2 for the learning and retrieval phases respectively.

Algorithm 1: Learning in TNN

input : $c,\ k,\ r,\ L$ & $\mathcal{S}$

initialization

$l = 2^k$,

Generate directed graph G with $n = c \times l$ nodes structured in c clusters of size l.

Assign clusters indices from 1 to L cyclically (similar to Figure 2)

begin

- **for** $s \in \mathcal{S}$ **do**
  - Activate the corresponding neurons to the sequence components;
  - Connect each active neuron to the consecutive r active neurons.

output: G

Algorithm 2: Winner Retrieval in TNN

input : G & $[s_1 : s_r]$

initialization

Activate r neurons in the first r clusters using $[s_1 : s_r]$

begin

- **for** $i \in [r+1 : L]$ **do**
  - Establish the output edges from previous r active neurons in the sequence;
  - Create a candidate set of nodes with maximum score in cluster i.
  - **if** *len(candidate set)* $== 1$ **then**
    - activate the only candidate node as winner and record it as s_i
  - **else if** *len(candidate set)* > 1 **then**
    - activate one of the candidate nodes randomly as winner and record it as s_i;

output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

For retrieval, the first r components of a previously learnt sequence, $[s_1 : s_r]$ and the learnt graph, G, are given and the complete sequence starting with $[s_1 : s_r]$ is expected.

In Algorithm 2, first each of the given r components are mapped to their related neurons in the first r clusters. Note that each component value is a number from 0 to $l - 1$. Then, the retrieval algorithm establishes the output edges from these r active neurons. The neurons in the destination cluster with highest input score will form the candidate set for the next component of the sequence. If there is just one candidate it will be added to the retrieved sequence and activated for retrieving the next component. Otherwise, the component must be chosen randomly among the candidates.

3.1.1 Winner-Cache Retrieval in TNN

In the case of Winner-Cache algorithm, the learning phase is similar, but the retrieval is more advanced. As reported in Algorithm 3, a temporary cache memory is used in the cases where random selection among winners results into an error which is detected later (see Figure 3 for an illustration).

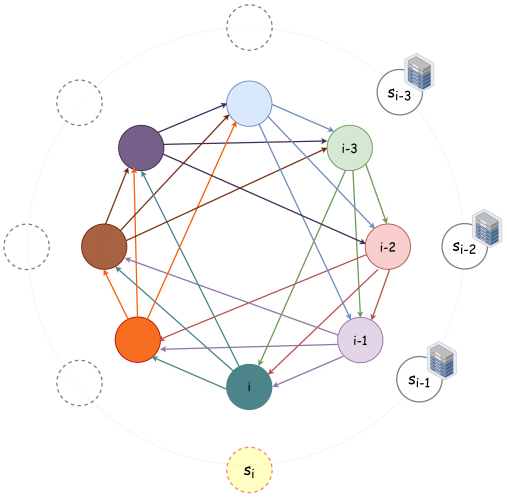


Figure 3: The mechanism of using temporary cache memory in the Winner-Cache retrieval is illustrated. The component s_i, with yellow color, represents the point in the retrieval where none of the nodes in cluster i has a score equal to $r = 3$ from last three previous activated neurons. This means that s_{i-1}, s_{i-2}, and s_{i-3} do not belong to any of previously stored sequences. Starting from cache memory in cluster $i - 3$ for component s_{i-3}, if there is an alternative candidate to be activated, we change the component, and start retrieving the sequence from that point. If in s_{i-3} the cache memory is empty, the algorithm checks for s_{i-2} and then s_{i-1}. At the end, if there is no alternative, or using the alternatives does not help, the candidate set for component s_i will be one of the winners, i.e. a node with maximum score.

The Cache-Winner algorithm proceeds as follows: whenever there is no unique

candidate, the component is chosen randomly among the candidates and other candidates will be recorded temporarily (up to assignment of the next r components). If the algorithm can not find a candidate connected to all the previous r active neurons, the algorithm starts retrieval from the earliest non-empty cache memory by randomly choosing another member. For the sake of brevity, we refer to this retrieval as *Cache* in the rest of paper.

Algorithm 3: Winner-Cache Retrieval in TNN.

input : G & $[s_1 : s_r]$

initialization

Activate r neurons in the first r clusters using $[s_1 : s_r]$

begin

$i = r$

while $i < L$ **do**

$i+ = 1$

Establish the output edges from last r active neurons in the sequence;

Create a candidate set of nodes with maximum score in cluster i.

if *maximum score* $< r$ **then**

search in the cache data of last r neurons ($[i - r : i - 1]$), find the first non-empty cache (j) and select a new member randomly.

Update the cache by removing the new member and start retrieval from that point (j) again by putting $i = j$.

if *len(candidate set)* $==$ *1* **then**

activate the only candidate node as winner and record it as s_i

else if *len(candidate set)* $>$ *1* **then**

activate one of the candidate nodes randomly as winner and record it as s_i;

Put the remaining members of the candidate set into a temporary cache;

keep the cached data until the next r neurons are assigned.

output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

3.1.2 Winner-Explore Retrieval for TNN

At this juncture, we introduce a retrieval technique which performs exploration within the forthcoming clusters to find a more accurate solution. As reported in Algorithm 4, whenever the candidate set in a cluster is not unique, by using the previous activated neurons, we produce possible candidates in the next clusters and consequently try to eliminate the current candidates by exploring the connections to the generated candidate

sets (see Figure 4 for an illustration). The maximum number of clusters that can be investigated ($r_{explore}$) is upper bounded by $r-1$. However, as will be discussed in section 4.1.1 one could limit the retrieval algorithm to explore shorter distances. For instance setting $r_{explore} < c-r$ in order to reach each cluster at most once for a specific component. Exploration involves searching for candidate sets in the following clusters and then trying to eliminate the number of candidates in the current cluster. The two techniques for this part are called Forward technique and Clique technique. In Forward technique, any candidate which is not connected to at least one node in the following clusters will be deleted from candidate set. Therefore, it is possible to find a unique candidate by reducing the size of candidate set. Clique technique is more advanced since it removes the candidates that are not in a tournament of largest possible size. We use term Clique for this technique to differentiate this technique from the learning on chain of tournaments.

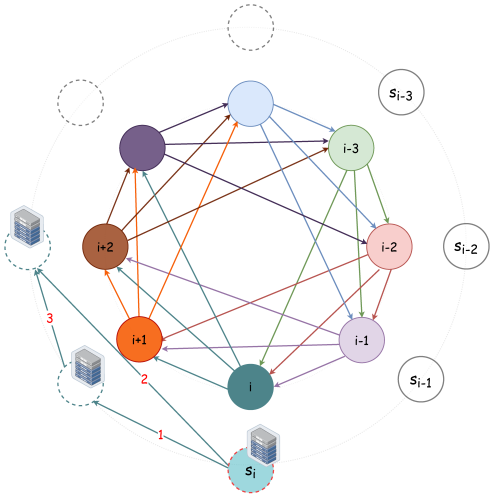


Figure 4: Using exploration technique to eliminate the number of components that are chosen randomly among the winners in Winner-Explore retrieval algorithm is illustrated. Suppose that by using the edges from $r=3$ previous nodes equivalent to s_{i-3}, s_{i-2}, and s_{i-1} to find s_i component, more than one option is found for the candidate set in cluster i. In this case, $r_{explore} = r-1 = 2$ previous components, i.e. s_{i-2} and s_{i-1} are used to create a candidate set in cluster $i+1$. In the Forward technique, the algorithm checks which candidates for component i are connected to at least one of the nodes in the candidate set in cluster $i+1$ (using links labeled with 1). If there is still more than one option, a candidate set in cluster $i+2$ will be constructed using s_{i-1}. Again, using Forward technique, the connections between candidates in cluster i and the candidate sets in the following $i+1$ and $i+2$ clusters are used to eliminate the options (labeled with 1 and 2). If still no unique option is available, Clique technique will be used which searches for the possible cliques of size 3 (using all the links labeled with $1, 2$, and 3). Since $r_{explore} = 2$, if there is no unique candidate in cluster i within the cliques, the process stops and the winner will be chosen randomly.

The retrieval process, as reported in Algorithm 4, searches for a candidate set in one cluster at each iteration: first by using the Forward technique, and then applying Clique technique. In the case of a non-unique option, algorithm proceeds by adding a new candidate set in the following cluster, and so on. The search for unique candidate stops whenever a unique option is found or all the clusters for exploration are taken into computation.

Algorithm 4: Winner-Explore Retrieval in TNN

input : G & $[s_1 : s_r], r_{explore}$

initialization

Activate r neurons in the first r clusters using $[s_1 : s_r]$

begin

for $i \in [r+1 : L]$ **do**

Establish the output edges from last r active neurons in the sequence;

Create a candidate set of nodes with maximum score in cluster i.

if *len(candidate set)* $==$ *1* **then**

activate the only candidate node as winner and record it as s_i

else if *len(candidate set)* $>$ *1* **then**

for $j \in [1 : r_{explore}]$ **do**

Create a candidate set in cluster $i+j$ using the $r-j$ activated nodes prior to i;

Construct a sub-graph of G with nodes of candidate sets in cluster i up to cluster $i+j$;

Update the candidate set in cluster i by keeping nodes with maximum output edges in sub-graph

if *len(candidate set)* $==$ *1* **then**

activate the only candidate node as winner and record it as s_i. `// Forward technique worked.`

else if *len(candidate set)* $>$ *1* **then**

Find all tournaments in the sub-graph including nodes from candidate set in cluster i with size $j+1$;

Update the candidate set in cluster i so that only candidates in such tournaments remain;

if *len(candidate set)* $==$ *1* **then**

activate the only candidate node as winner and record it as s_i. `// Clique technique worked.`

else if *len(candidate set)* $==$ *0* ***or*** $j == r-1$ **then**

Return the last non-empty candidate set as the final candidate set for cluster i;

output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

3.2 Feedback TNN Structure

In this structure, the learning phase sets tournaments with forward and backward connections. Each node in a tournament of size $r+1$, has r_{fwd} links to the forthcoming clusters and receives r_{fbk} links from the forthcoming $[r_{fwd}+1 : r]$ active neurons, where $0 \geq r_{fbk} \geq r_{fwd}$ and $r = r_{fwd} + r_{fbk}$ (see Figure 5).The original TNN can be seen as a Feedback TNN with zero feedback links ($r_{fbk} = 0$).

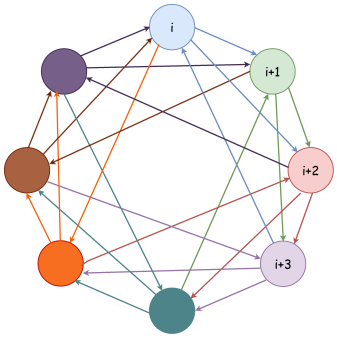


Figure 5: In the chain of tournament structure with feedback connections, the first r_{fwd} connections of each tournament are clockwise and the next r_{fbk} connections are counterclockwise. In this illustration, $r_{fwd} = 2$ and $r_{fbk} = 1$.

For storing sequence $s \in \mathcal{S}$, where $s = s_1 s_2 \cdots s_L$, the clockwise connections in the network will be as follows:

$$n_{ij} \rightarrow n_{i'j'}, \text{ if: } \begin{cases} f_i(s_{i+(\pi-1)c}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{ and, } 1 \leq \delta_i(i') \leq r_{fwd}, \tag{3}$$

and for counterclockwise connections:

$$n_{ij} \leftarrow n_{i'j'}, \text{ if: } \begin{cases} f_i(s_{i+(\pi-1)c}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{ and, } r_{fwd} \leq \delta_i(i') \leq r \tag{4}$$

where $1 \leq \pi \leq \lfloor \frac{L}{c} \rfloor$. In general, for $s \in \mathcal{S}$, if the above conditions are satisfied for a given π such that $n_{ij} \rightarrow n_{i'j'}$, we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 1$. Similarly we set $N_{s,\pi}(n_{i'j'}, n_{ij}) = 1$, if $n_{ij} \leftarrow n_{i'j'}$; otherwise we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 0$, and $N_{s,\pi}(n_{i'j'}, n_{ij}) = 0$.

At the end of learning or storing process, the network has the following connections:

$$\mathcal{W} = \{\omega_{(ij)(i'j')} | \text{ if } \exists\, s \in \mathcal{S}, \text{ and } \exists\, \pi \in [1 : \lfloor \frac{L}{c} \rfloor] \text{ s.t. } N_{s,\pi}(n_{ij}, n_{i'j'}) = 1\} \tag{5}$$

where $\omega_{(ij)(i'j')}$ is a directed edge from n_{ij} to $n_{i'j'}$ and $1 \leq i, i' \leq c,\ 1 \leq j, j' \leq l$ (see Algorithm 5 for the learning process). In Figure 5, activated neurons in cluster i are connected to the activated neurons in clusters $i+1$ and $i+2$ clockwise, whereas

activated neurons in cluster $i+3$ are connected to the activated neurons in cluster i counterclockwise.

Algorithm 5: Learning in Feedback TNN

input : $c,\ k,\ r,\ r_{fwd},\ L$ & $\mathcal{S}$

initialization

$l = 2^k,\ r_{fbk} = r - r_{fwd}$

Generate directed graph G with $n = c \times l$ nodes structured in c clusters of size l.

Assign clusters indices from 1 to L cyclically (see Figure 5 for labeling)

begin

  for $s \in \mathcal{S}$ **do**

    Activate the corresponding neurons to the sequence;

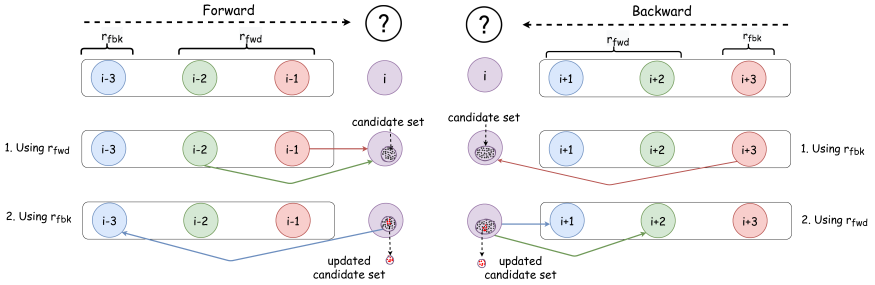
    Connect each active neuron (say in cluster i) to the active neurons in the next r_{fwd} clusters ($[i+1 : i+r_{fwd}]$);

    Connect each active neuron to the previous r_{fbk} active neurons in clusters $[i-r : i-r_{fwd}-1]$;

output: G

3.2.1 Retrieval in Feedback TNN

Here we introduce two retrieval algorithms, Feedback-Forward (Algorithm 6) and Feedback-Backward (Algorithm 7), which can retrieve a complete sequence from any given segment. To do so, we need a pre-matching process to find the clusters on which the given sequence segment was stored (see Figure 6 for an illustration of Feedback-Forward and Feedback-Backward processes).



(a) For Forward retrieval in Feedback TNN, first a candidate set in cluster i is created using the connections from active neuron in clusters $i-1$ and $i-2$ (since $r_{fwd}=2$). If there is a unique winner candidate, the algorithm stops, otherwise a sub-graph is constructed with the candidate set and the active neuron in cluster $i-3$ (since $r_{fbk}=1$). The candidate set will be updated by keeping nodes with maximum score.

(b) For Backward retrieval in Feedback TNN, first a candidate set in cluster i is created using the connections from active neuron at cluster $i+3$ (since $r_{fbk}=1$). If there is a unique winner candidate, the algorithm stops, otherwise a sub-graph is constructed with the candidate set and the active neurons in clusters $i+1$ and $i+2$ (since $r_{fwd}=2$). The candidate set will be updated by keeping nodes with maximum score.

Figure 6: Consider the structure in Figure 5 where $r_{fwd}=2$ and $r_{fbk}=1$. Given a segment of $r=3$ components, Forward and Backward retrieval processes are illustrated respectively in (a) and (b).

Feedback-Forward algorithm (hereafter Forward) retrieves the sequence given the first r components of it. This retrieval is performed in two phases: first, by using the r_{fwd} connections, and then if the winning candidate is not unique, the r_{fbk} connections are used to eliminate the number of candidates, as reported in Algorithm 6.

Feedback-Backward algorithm (hereafter Backward), retrieves the sequence given the last r components of a sequence. As reported in Algorithm 7, the algorithm first uses the r_{fbk} input edges to make an initial candidate set, and then the output edges from the candidate set is used to eliminate the number of candidates.

Algorithm 6: Feedback-Forward Retrieval in Feedback TNN

input : G & $[s_1 : s_r]$

initialization

Activate r neurons in the first r clusters using $[s_1 : s_r]$

Assign clusters indices from 1 to L cyclically

begin

for $i \in [r+1 : L]$ **do**

Establish the output edges from previous r_{fwd} active neurons in the sequence;

Create a candidate set of nodes with maximum score in cluster i.

if *len(candidate set)* $==$ *1* **then**

activate the only candidate node as winner and record it as s_i

else if *len(candidate set)* $>$ *1* **then**

A sub-graph of G with nodes from candidate set in cluster i, and previous r_{fbk} active neurons in clusters $[i-r : i-r_{fwd}]$ is constructed;

The new candidate set for cluster i is updated by keeping the nodes which have maximum output edges in the sub-graph;

Select one node from the updated candidate set as winner and record it as s_i;

output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

Algorithm 7: Feedback-Backward retrieval in Feedback TNN.

input : G & $[s_{L-r+1} : s_L]$

initialization

Activate r neurons in the related r clusters using $[s_{L-r+1} : s_L]$

Assign clusters indices from 1 to L cyclically

begin

for $i \in [L - r : 1; -1]$ **do**

Establish the output edges from r_{fdk} active neurons in clusters $[i + r_{fwd} : i + r]$;

Create a candidate set of nodes with maximum score in cluster i.

if *len(candidate set)* $==$ *1* **then**

activate the only candidate node as winner and record it as s_i

else if *len(candidate set)* $>$ *1* **then**

A sub-graph of G with nodes from candidate set in cluster i, and the next r_{fwd} active neurons is constructed;

The new candidate set for cluster i is updated by keeping the nodes with maximum score (maximum output edges) in the sub-graph;

Select one node from the updated candidate set as winner and record it as s_i;

output: $s_{[s_{L-r+1}:s_L]}$ `// Retrieved sequence given` $[s_{L-r+1} : s_L]$

Note that Winner (Algorithm 2) can be seen as a special case of Forward (Algorithm 6) when $r_{fwd} = r$ and $r_{fbk} = 0$. In Figure 6a, only the first step that uses r_{fwd} is applicable. On the other hand, in the case of the original TNN, the Backward algorithm starts with a candidate set of size l and makes a sub-graph with the given $r_{fwd} = r$ components, since $r_{fbk} = 0$ and there is no input connection. In Figure 6b, only the second step that uses r_{fwd} is applicable.

3.3 Error Types

Based on the argument of Jiang et al. (2016), two different error types could be distinguished; an error type that is due to prior retrieval errors in simulation, and an error type that is structural and which is caused by an excessive network density. The structural error type could happen even if all the previous r components are given correctly.

Component Error Rate (CER) and Sequence Error Rate (SER) address the simulation error; CER is defined as the ratio of the number of incorrect components over the number of total retrieved components, whereas SER is defined as the number of sequences that are failed to be retrieved correctly over the total number of sequences.

Structural Component Error Rate (S-CER) and Structural Sequence Error Rate (S-SER) address the structural error. According to Jiang et al. (2016), the S-CER can be estimated as the error rate at a single retrieval step when the provided previous r components are correct.

$$P_{S-CER} = 1 - (1 - d^r)^{l-1} \tag{6}$$

where d is the network density which is the ratio of number of established connections during the storage process over all possible connections that the network structure allows. The density is calculated (in Jiang et al., 2016, equation 7) as:

$$d = 1 - \left(1 - \frac{1}{l^2}\right)^{\frac{|S|L}{c}} \tag{7}$$

At the sequence level, S-SER is estimated (in Jiang et al., 2016, equation 9) as:

$$P_{S-SER} = 1 - (1 - d^r)^{(l-1)(L-r)} \tag{8}$$

Please note that the density in the Feedback TNN structure is the same as the density of the original TNN structure (equation 7). This is due to the fact that the density is calculated based on the probability of having a connection between two nodes, and in the case of Feedback TNN just the directions of some connections are changed while their number remains the same. Moreover, based on the definition of structural errors, equations 6 and 8 are valid for Cache, Explore and Feedback TNN retrievals.

4 Simulation Results

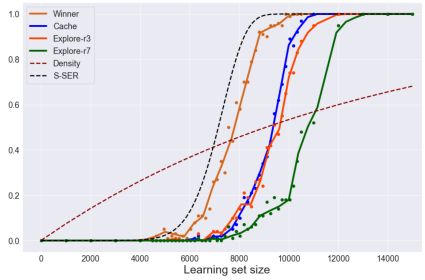
In this section, the simulation results for different algorithms are presented in order to show the robustness of storage and to compare different structures. Learning processes for TNN and Feedback TNN structures (Algorithm 1 and Algorithm 5, respectively) are considered when $c = 20$, $k = 8$, $l = 2^8 = 256$, $r = 12$, $r_{fwd} = 6$, $r_{fbk} = r - r_{fwd} = 6$, and $L = 100$. Regarding the retrieval, four scenarios; *Winner* (Algorithm 2), *Cache* (Algorithm 3), *Explore* (Algorithm 4), *Forward* (Algorithm 6), and *Backward* (Algorithm 7) are simulated and compared.

The sequences in the learning set are different in at least one of the first r components. For instance, a learning set of size 1000 is a set of 1000 sequences that all are different in at least one component in the 12 first components. To see if the memorized sequences can be retrieved, 100 of the learnt sequences are randomly chosen from each learning set. To reduce randomness effect, we fixed the 100 choices of sequences in the learning set of each size (varies between 10 to 15000), in simulations for all the retrieval algorithms.

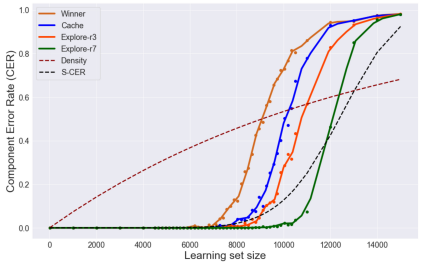
4.1 TNN Retrieval Results

Figure 7 depicts the error rate for a range of learning set sizes, for different retrieval algorithms, namely, Winner (Algorithm 2), Cache (Algorithm 3) , Explore with $r_{explore} = 3$, and $r_{explore} = 7$ (Algorithm 4). To illustrate the power of the algorithms with respect

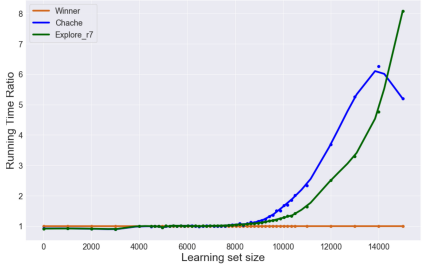
to the structure of the network, the calculated density and structured error are also plotted. It is clear from the results that retrieval with the exploration when $r_{explore} = 7$ is far better than the rest of scenarios. For instance, when the learning set is composed of 10000 sequences, each of size 100, the SER (Figure 7a) for the Winner is one, which means that no sequence can be retrieved correctly with the original algorithm. While this value is about 0.7 for the algorithm with cache memory and about 0.6 when the exploration technique is used with $r_{explore} = 3$, and the SER for exploration with $r_{explore} = 7$ is less than 0.2. This superiority of exploration algorithm can easily be tracked in the CER results (Figure 7b). For instance, for the same learning set, the CER for Winner is 0.75, for Cache it is 0.4, for Explore with $r_{explore} = 3$ it equals to 0.3, and for Explore with $r_{explore} = 7$ it is near zero.



(a) Sequence error rate (SER)



(b) Component error rate (CER)



(c) Running time ratio for Explore-$r7$ and Cache over Winner.

Figure 7: Comparison between retrieval algorithms on the TNN structure; Winner (Algorithm 2), Cache (Algorithm 3) , Explore with $r_{explore} = 3$, and $r_{explore} = 7$ (Algorithm 4). The running time ratios of Explore (with $r_{explore} = 7$) and Cache over Winner are reported in 7c.

In Figure 7a, the simulated error value for all retrieval methods are less than S-SER which is obtained from equation 8. This can be explained by the fact that the S-SER error estimation is based on the probability of having at least two nodes in a cluster that all the previous r components are connected to. In this case, for the simplest version of retrieval algorithms, Winner, one candidate will be chosen randomly. In other words,

S-SER is an upper bound for SER and in the case that all the choices are unique (S-SER $= 0$), there will be no error (SER $= 0$). Although there is no guarantee that the randomly chosen candidate is the desired one, the SER value is slightly less than the S-SER. Obviously, the more sophisticated retrieval algorithms, Cache and Explore, reduce the random selections and therefore, the number of errors. The structure error is a function of network density and as can be seen in Figure 7, higher density leads to higher structure error.

For S-CER (Figure 7b), the argument is different and the simulated error values in retrieval process are higher than S-CER. To calculate S-CER, the assumption is that the previous retrieved components are correct and S-CER estimates the probability of having at least two nodes that are fully connected to the previous r components. However, in the simulation, the values of some of r previous components are faulty and as a result the decision is not based on correct components. Therefore, in a sequence retrieval, errors at each component could be propagated to the rest of retrieval and simulated error CER will be higher than S-CER which assumes the r components are correct.

The reported results in Figure 7 suggest Explore retrieval with higher number of steps. Cache algorithm is also promising, but for large learning sets it has a low speed. When the network density increases, Cache retrieval process creates larger candidate sets for each component and therefore larger cache memory, and the algorithm might go through all the options to find the correct component. Explore, on the other hand, must explore longer distances that is the source of complexity in Explore. Figure 7c compares the simulation running time between Explore-$r7$ and Cache with Winner for different learning set sizes. The running time up to a learning set size of 8000 for all the three algorithms is the same, while Explore-$r7$ and Cache perform far better than Winner; compare the low performance of Winner ($SER = 0.52$) with the performance of Cache ($SER = 0.08$) and Explore-$r7$ ($SER = 0.02$). As another example, for learning set size 10000, $SER = 1$ for Winner; while Explore-$r7$ has $SER = 0.18$ and running time ratio 1.2, and Cache has $SER = 0.86$ and running time ratio 1.7.

This shows that for reasonable error values (say less than 0.1), the running time ratio is at the same level of Winner in both cases. Interestingly, the running time for Cache reaches a peak for a learning set of size 14000 and thereafter starts to decline for larger learning sets as shown in Figure 7c. This can be explained by the excessive density so that the probability of having full score candidate at each step increases and therefore the algorithm can not detect an error which reduces the processing time for checking the Cache memory.

In Figure 7b, only Explore algorithm with $r_{explore} = 7$ that investigates further clusters shows lower error than S-CER until the density about 0.6 and learning set of size 12000. We will have a closer look at the simulation results for the Explore algorithm below.

4.1.1 More Investigation on Explore Retrieval Algorithm

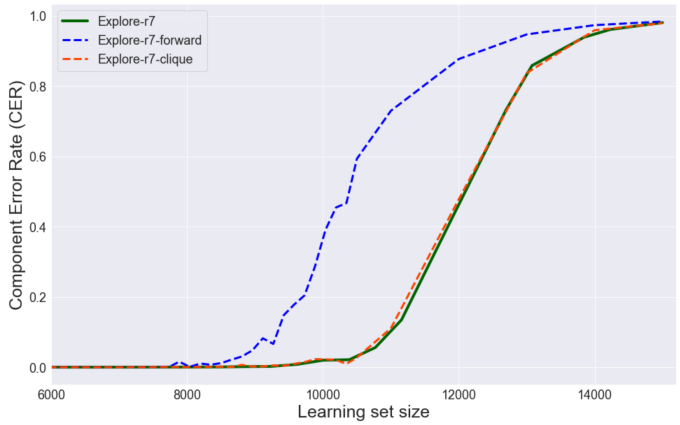
In Explore retrieval, by starting from distance one, the algorithm uses Forward and Clique techniques consecutively and increases the exploration distance until a unique candidate is found or $r_{explore}$ limit is met. Clique technique is more powerful but it is more computationally expensive than Forward technique. Figure 8a shows that by

using the Clique technique alone (red dashed line) the exploration performance does not change, whilst Forward technique alone (blue dashed line) is far less effective than the achieved results by exploration algorithm. This is an expected result since Clique technique is endowed with Forward technique.

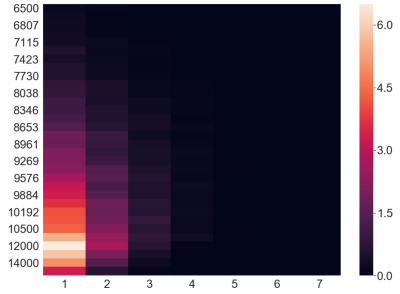
Figure 8b and 8c show the number of components that Forward and Clique techniques successfully retrieved (unique winner), respectively in the course of retrieving each sequence. The columns show the exploration distance and the rows show the size of learning sets. It is noteworthy that the first column in Figures 8c is all zero since for a distance one, a forward connection and a tournament of size 2 are the same, and the Forward technique is prior to the Clique technique in Algorithm 6.

As reported in Figure 7b, the Winner handles the retrieval when the learning set sizes are up to 7000. Until this point, no exploration is demanded. But with larger sizes of learning set and whenever it comes to the exploration phase, most of the cases can be retrieved with exploration of distance one. This, however, does not mean that the best choice, in terms of time/accuracy trade off, is $r_{explore} = 1$. When the size of learning sets gets higher, the Clique technique gets more involved. Because the higher sizes of candidate sets in under exploration clusters increases the searching domain, which results Forward technique to be failed in retrieval and Clique technique starts to retrieve. Let us consider for instance the learning set sizes around $12000 - 13000$ which is the highest number of successful retrievals per sequence using Explore retrieval (Figure 7a). For these sizes the CER error is high, for example it is about 0.46 for learning set of size 12000 and equals 0.85 when the learning set size is 13000 and therefore the overall retrieval is not successful. Interestingly, the S-CER also beats CER at around 12000 (Figure 7b) which shows that high density can not be managed with exploration technique as well.

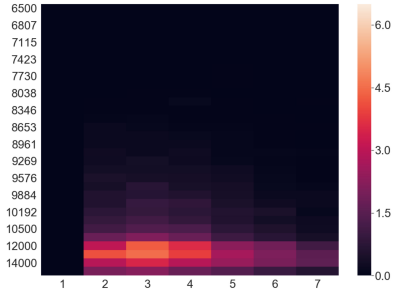
For learning sets of size 11000, the CER for Explore-$r7$ is 0.074 (Figure 7b) while without exploration technique the CER value equals one for learning set sizes larger than 10000. Figure 8b and 8c show decrease in the successful cases at exploration with higher distances, say 6 or 7 which suggests that extra exploration is not worth the computation. We found $r_{explore} = 7$ as a suitable choice for this setting of parameters.



(a) CER for Explore algorithm compared with the cases that either Forward technique or Clique technique is used.



(b) Number of unique winner components which are found at $[1:7]$ exploration distances using Forward technique.



(c) Number of unique winner components which are found at $[1:7]$ exploration distances using Clique technique (with tournament sizes $[1:7]+1$)

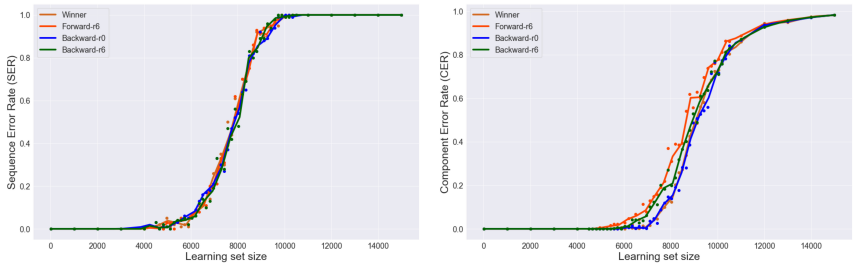
Figure 8: Analysis of Explore retrieval; Forward technique vs. Clique technique and the required exploration distance for finding a unique component. Results of learning set sizes between 6000 and 15000 are depicted.

4.2 Feedback TNN Retrieval Results

Figure 9 shows the retrieval error of Feedback TNN learning when $r = 12$ & $r_{fwd} = 6$ (Forward-$r6$ and Backward-$r6$) together with the retrieval error of original learning method (TNN) with Winner and Backward-$r0$ retrievals when $r = 12$. We start the Winner and Forward-$r6$ retrievals when the first $r = 12$ components are given, and Backward-$r6$ and Backward-$r0$ when the last $r = 12$ components are given.

Figure 9a confirms that the sequence retrieval results in Feedback TNN can be as

accurate as the original TNN memories. It is almost the same for CER (Figure 9b), however the results for the original TNNs are slightly better. We can explain this as a result of errors in recent previous $r_{fwd} = 6$ components. Consider the case that the algorithm finds a unique candidate for the current component based on last $r_{fwd} = 6$ components, without considering the other $r_{fbk} = 6$ links, and selects it as the only winner, while it can be incorrect candidate due to some errors in previous steps. However, if the algorithm uses all the r_{fwd} and r_{fbk} links the candidate set might composed of more components, which are not necessarily of full score. In this case, the final candidate will be chosen randomly, and therefore there is a chance of correct component selection. The above argument could similarly explain why CER for Backward-$r0$ are slightly better than Backward-$r6$. Note that the errors in Feedback TNN retrievals might cause more random choices in retrieval of the rest of components (see section 4.3 for an analysis of randomly chosen components). Indeed, such errors do not increase SER but CER could be affected as seen in Figure 9b.



(a) Sequence error rate

(b) Component error rate

Figure 9: Comparison between the original TNN learning method and the learning in Feedback TNN using Winner, Forward and Backward retrievals.

In summary, in Feedback TNN the retrieval is faster than TNN, the SER performance is the same for both, but TNN could be slightly better in CER performance.

4.3 Randomness in Simulated Retrievals; an Overall Look

Figure 10 provides a general overview on the number of cases in average that retrieval algorithms select the final component randomly from the candidate set. The success in policy of reducing the number of cases with random decision in Cache and Explore retrievals to achieve better retrieval performance is clearly shown in the last three columns related to these retrievals. For instance, when the learning set size equals 11000, nearly 50 components out of $L - r = 88$ are chosen randomly for Winner, as the original retrieval algorithm, but it is about 20 for Cache, 15 for Explore with $r_{explore} = 3$, and almost zero for Explore with $r_{explore} = 7$. The number of random choices for Feedback TNN structure, both Forward and Backward, is slightly higher than Winner and Backward-$r0$. The argument is that the errors that appear due to the wrong unique retrieval, produce more error afterwards in the sequence, and therefore more random

winner retrieval cases in total. We also can observe a slightly higher number of random winner selection in the Backward-$r0$. This could be related to the learning set generation in our simulations. The sequences in a learning set, are forced to be different in at least one of the first r components. Therefore, the Winner can start the retrieval with the unique sequence, while in the Backward-$r0$ more than one sequence can match with the given last r components.

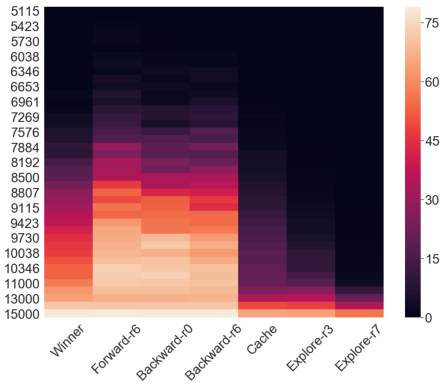


Figure 10: A comparison between number of random selection of winner candidate in different scenarios.

5 Discussion and Concluding Remarks

In this study, two-fold contributions within the field of TNN structures were presented; first, we proposed a more general learning and retrieval structure called Feedback TNN, and second, we devised two more accurate retrieval algorithms in comparison with the Winner algorithm.

In Feedback TNN, each segment of sequence of length $r + 1$ is mapped into a tournament in $r + 1$ consecutive clusters where each neuron has r_{fbk} input edges and $r_{fwd} = r - r_{fbk}$ output edges. The proposed retrieval for the Feedback TNN operates in two phases, in a faster manner than TNN retrieval, and generates the same sequence error rate while producing a slightly weaker component error rate.

The original TNN can be considered as a special case of Feedback TNN with zero feedback connections. Using feedback connections, we obtained results of sequence retrieval as precise as the original structure, with the possibility of faster retrieval. One might also divide the r forward connections into two parts, say r_1 and r_2, and try to retrieve the component using the most recent r_1 active neurons, and if it is not possible to uniquely retrieve, use the rest of r_2 neurons. More generally, one can try to retrieve by starting from the last active neuron and reduce the size of the candidate set (losers-kicks-out), and adding more active neurons to the retrieval process, until either one winner candidate remains or all the r active neurons are used.

By introducing Backward retrieval in this paper, we showed that it is possible to

get a part of a sequence, no matter its location, and retrieve the rest. In this case, the retrieval algorithm must be able to first locate a tournament matching the given sub-sequence, and later retrieve the whole sequence from both directions. Backward retrieval is compatible with both TNN and Feedback TNN structures, but Feedback TNN with non-zero feedback links is preferable since the Backward retrieval algorithm can start with a smaller size candidate set.

In order to improve the retrieval accuracy for a given network, we suggested two algorithms with the overall strategy to limit the number of random selections during retrieval. The Cache retrieval (Algorithm 3) uses a temporary cache memory for the last r components to record the candidate set of winners whenever the chosen winner is not unique. These cached alternatives are used whenever the algorithm detects an error by observing no candidate having a full score. The reported results in section 4 confirm the usefulness of this method. The more advanced, and successful, retrieval algorithm (Algorithm 4) explores the forthcoming clusters to find a unique candidate in the current cluster. This algorithm somehow investigates the consequence of choosing each candidate by checking its connections to the possible future components and decides more judiciously. This algorithm produces the best results.

Explore-Winner is a more reliable retrieval method than Cache-Winner since it limits the number of random choices using the data in the forthcoming clusters, while Cache-Winner tries to correct the errors by testing other possibilities. Cache-Winner might be computationally expensive in higher densities where candidate sets of winners are larger and therefore, larger sets are cached. Finding an optimal $r_{explore}$, for exploration distance limit, as shown in section 4.1.1, is a trade-off between time and accuracy. Although not reported in the simulations, both Cache-Winner and Explore-Winner can be used in Feedback TNN and for Backward retrieval.

Similar to the double-layer structure proposed by Jiang et al. (2016), it is possible to consider a hierarchical structure by adding an extra connectivity level. Moreover, similar to the technique used in (Mofrad et al., 2016) a precoding could dramatically increase the storage and retrieval capacity by forcing patterns to be well separated and therefore reducing the common tournaments in different patterns.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback who helped to improve the quality of the manuscript. The source code of the simulations is made publicly available online under this link: `https://github.com/Asieh-A-Mofrad/Tournament-Based-Sequence-Storage`.

References

Aboudib, A., Gripon, V., & Coppin, G. (2016). A neural network model for solving the feature correspondence problem. In *International Conference on Artificial Neural Networks* (pp. 439–446).: Springer.

Aboudib, A., Gripon, V., & Jiang, X. (2014). A study of retrieval algorithms of sparse

messages in networks of neural cliques. In *COGNITIVE 2014: the 6th International Conference on Advanced Cognitive Technologies and Applications* (pp. 140–146).

Aliabadi, B. K., Berrou, C., Gripon, V., & Jiang, X. (2014). Storing sparse messages in networks of neural cliques. *IEEE Transactions on neural networks and learning systems*, 25(5), 980–989.

Berrou, C., Dufor, O., Gripon, V., & Jiang, X. (2014). Information, noise, coding, modulation: What about the brain? In *Turbo Codes and Iterative Information Processing (ISTC), 2014 8th International Symposium on* (pp. 167–172).: IEEE.

Berrou, C. & Gripon, V. (2010). Coded hopfield networks. In *Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on* (pp. 1–5).: IEEE.

Berrou, C. & Kim-Dufor, D.-H. (2018). A connectionist model of reading with error correction properties. In *Human Language Technology. Challenges for Computer Science and Linguistics: 7th Language and Technology Conference, LTC 2015, Poznań, Poland, November 27-29, 2015, Revised Selected Papers*, volume 10930 (pp. 304).: Springer.

Boguslawski, B., Gripon, V., Seguin, F., & Heitzmann, F. (2014). Huffman coding for storing non-uniformly distributed messages in networks of neural cliques. In *AAAI 2014: the 28th Conference on Artificial Intelligence*, volume 1 (pp. 262–268).

Brea, J., Senn, W., & Pfister, J.-P. (2011). Sequence learning with hidden units in spiking neural networks. In *Advances in neural information processing systems* (pp. 1422–1430).

Danilo, R., Jarollahi, H., Gripon, V., Coussy, P., Conde-Canencia, L., & Gross, W. J. (2015). Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 2501–2504).: IEEE.

Gripon, V. & Berrou, C. (2011). Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, 22(7), 1087–1096.

Gripon, V. & Berrou, C. (2012). Nearly-optimal associative memories based on distributed constant weight codes. In *Information Theory and Applications Workshop (ITA), 2012* (pp. 269–273).: IEEE.

Gripon, V., Heusel, J., Löwe, M., & Vermet, F. (2016). A comparative study of sparse associative memories. *Journal of Statistical Physics*, 164(1), 105–129.

Hacene, G. B., Gripon, V., Farrugia, N., Arzel, M., & Jezequel, M. (2017). Finding all matches in a database using binary neural networks. *COGNTIVE 2017*, (pp.67).

Hacene, G. B., Gripon, V., Farrugia, N., Arzel, M., & Jezequel, M. (2019). Budget restricted incremental learning with pre-trained convolutional neural networks and binary associative memories. *Journal of Signal Processing Systems*, 91(9), 1063–1073.

Hawkins, J. & Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in neural circuits*, 10, 23.

Hawkins, J. & Blakeslee, S. (2007). *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan.

Hawkins, J., George, D., & Niemasik, J. (2009). Sequence memory for prediction, inference and behaviour. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 364(1521), 1203–1209.

Hoffmann, H. (2019). Sparse associative memory. *Neural computation*, 31(5), 998–1014.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8), 2554–2558.

Hopfield, J. J. (2008). Searching for memories, sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. *Neural Computation*, 20(5), 1119–1164.

Jarollahi, H., Gripon, V., Onizawa, N., & Gross, W. J. (2015). Algorithm and architecture for a low-power content-addressable memory based on sparse clustered networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(4), 642–653.

Jarollahi, H., Onizawa, N., Gripon, V., & Gross, W. J. (2014). Algorithm and architecture of fully-parallel associative memories based on sparse clustered networks. *Journal of Signal Processing Systems*, 76(3), 235–247.

Jiang, X. (2014). *Storing sequences in binary neural networks with high efficiency*. PhD thesis, Télécom Bretagne, Université de Bretagne Occidentale.

Jiang, X., Gripon, V., Berrou, C., & Rabbat, M. (2016). Storing sequences in binary tournament-based neural networks. *IEEE transactions on neural networks and learning systems*, 27(5), 913–925.

Jiang, X., Marques, M. R. S., Kirsch, P.-J., & Berrou, C. (2015). Improved retrieval for challenging scenarios in clique-based neural networks. In *International Work-Conference on Artificial Neural Networks* (pp. 400–414).: Springer.

Kim, D.-H., Park, J., & Kahng, B. (2017). Enhanced storage capacity with errors in scale-free hopfield neural networks: An analytical study. *PloS one*, 12(10), e0184683.

Krotov, D. & Hopfield, J. J. (2016). Dense associative memory for pattern recognition. In *Advances in neural information processing systems* (pp. 1172–1180).

Larras, B., Chollet, P., Lahuec, C., Seguin, F., & Arzel, M. (2018). A fully flexible circuit implementation of clique-based neural networks in 65-nm cmos. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(5), 1704–1715.

Larras, B. & Frappé, A. (2020). On the distribution of clique-based neural networks for edge ai. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*.

Marques, M. R. S., Hacene, G. B., Lassance, C. E. R. K., & Horrein, P.-H. (2017). Large-scale memory of sequences using binary sparse neural networks on gpu. In *2017 International Conference on High Performance Computing & Simulation (HPCS)* (pp. 553–559).: IEEE.

Maurer, A., Hersch, M., & Billard, A. G. (2005). Extended hopfield network for sequence learning: Application to gesture recognition. In *International Conference on Artificial Neural Networks* (pp. 493–498).: Springer.

Mofrad, A. A., Ferdosi, Z., Parker, M. G., & Tadayon, M. H. (2015). Neural network associative memories with local coding. In *Information Theory (CWIT), 2015 IEEE 14th Canadian Workshop on* (pp. 178–181).: IEEE.

Mofrad, A. A. & Parker, M. G. (2017). Nested-clique network model of neural associative memory. *Neural Computation*.

Mofrad, A. A., Parker, M. G., Ferdosi, Z., & Tadayon, M. H. (2016). Clique based neural associative memories with local coding and pre-coding. *Neural Computation*, 28, 1–21.

Olshausen, B. A. & Field, D. J. (2004). Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4), 481–487.

Rinkus, G. J. (2010). A cortical sparse distributed coding model linking mini-and macrocolumn-scale functionality. *Frontiers in neuroanatomy*, 4, 17.

Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature*.

Yao, Z., Gripon, V., & Rabbat, M. (2014). A gpu-based associative memory using sparse neural networks. In *2014 International Conference on High Performance Computing & Simulation (HPCS)* (pp. 688–692).: IEEE.

Appendix A

A.1 Neural network associative memories with local coding

Asieh Abolpour Mofrad, Zahra Ferdosi, Matthew G. Parker, and Mohammad H. Tadayon

Mofrad, A. A., Ferdosi, Z., Parker, M. G., & Tadayon, M. H. (2015, July). Neural network associative memories with local coding. In 2015 IEEE 14th Canadian Workshop on Information Theory (CWIT) (pp. 178-181). IEEE.

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230844618 (print)
9788230841280 (PDF)