A Non-Binary Associative Memory with Exponential Pattern Retrieval Capacity and Iterative Learning

Amir Hesam Salavati†, K. Raj Kumar‡, and Amin Shokrollahi†
†:Laboratoire d'algorithmique (ALGO), Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland
E-mail: {hesam.salavati,amin.shokrollahi}@epfl.ch
‡:Qualcomm Research India, Bangalore - 560066, India
E-mail: kumarraj@qti.qualcomm.com

***Abstract*—We consider the problem of neural association for a network of non-binary neurons. Here, the task is to first memorize a set of patterns using a network of neurons whose states assume values from a finite number of integer levels. Later, the same network should be able to recall previously memorized patterns from their noisy versions. Prior work in this area consider storing a finite number of *purely random* patterns, and have shown that the pattern retrieval capacities (maximum number of patterns that can be memorized) scale only linearly with the number of neurons in the network.**

In our formulation of the problem, we concentrate on exploiting redundancy and internal structure of the patterns in order to improve the pattern retrieval capacity. Our first result shows that if the given patterns have a suitable linear-algebraic structure, i.e. comprise a sub-space of the set of all possible patterns, then the pattern retrieval capacity is in fact exponential in terms of the number of neurons. The second result extends the previous finding to cases where the patterns have weak minor components, i.e. the smallest eigenvalues of the correlation matrix tend toward zero. We will use these minor components (or the basis vectors of the pattern null space) to both increase the pattern retrieval capacity and error correction capabilities.

An iterative algorithm is proposed for the learning phase, and two simple algorithms are presented for the recall phase. Using analytical methods and simulations, we show that the proposed methods can tolerate a fair amount of errors in the input while being able to memorize an exponentially large number of patterns.

***Index Terms*—Neural associative memory, Error correcting codes, Message passing, Stochastic learning, Dual-space method**

I. Introduction

Neural associative memory is a particular class of neural networks capable of memorizing (learning) a set of patterns and recalling them later in presence of noise, i.e. retrieve the correct memorized pattern from a given noisy version. Starting from the seminal work of Hopfield in 1982 [1], various artificial neural networks have been designed to mimic the task of the neuronal associative memory (see for instance [2], [3], [4], [5], [6]).

In essence, the neural associative memory problem is very similar to the one faced in communication systems where the goal is to reliably and efficiently retrieve a set of patterns (so called "codewords") form noisy versions. More interestingly, the techniques used to implement an artificial neural associative memory looks very similar to some of the methods used in graph-based modern codes to decode information. This makes the pattern retrieval phase in neural associative memories very similar to iterative decoding methods in modern coding theory.

However, despite the similarity in the task and techniques employed in both problems, there is a huge gap in terms of efficiency. Using binary codewords of length n, one can construct codes that are capable of reliably transmitting 2^{rn} codewords over a noisy channel, where $0 < r < 1$ is the code rate [7]. The optimal r (i.e. the largest value that permits the almost sure recovery) depends on the noise characteristics of the channel and is known as the Shannon capacity [8]. In fact, the Shannon capacity is achievable in certain cases, for example by LDPC codes over AWGN channels.

In current neural associative memories, however, with a network of size n one can only memorize $O(n)$ binary patterns of length n [9], [2]. To be fair, it must be mentioned that these networks are designed such that they are able to memorize any possible set of *randomly* chosen patterns(e.g., [1], [2], [3], [4]). Therefore, although humans cannot memorize random patterns, these methods provide artificial neural associative memories with a pleasant sense of generality.

However, this generality severely restricts the efficiency of the network since even if the input patterns have some internal redundancy or structure, current neural associative memories could not exploit this redundancy in order to increase the number of memorizable patterns or improve error correction during the recall phase. In fact, concentrating on redundancies within patterns is a fairly new viewpoint, which is in harmony with coding techniques where one designs codewords with certain degree of redundancy and then use this redundancy to correct corrupted signals at the receiver's side.

In this paper, we focus on bridging the performance gap between the coding techniques and neural associative memories. Our proposed neural network exploits the inherent structure of the input patterns in order to increase the pattern retrieval capacity from $O(n)$ to $O(a^n)$, where $a > 1$. More specifically, the proposed neural network is capable of learning and reliably recalling given patterns when they come from a subspace with dimension $k < n$ of all possible n-dimensional patterns. Thus, although the model does not have the versatility of the traditional associative memories, the capacity is boosted by a great extent. Furthermore, traditional associative memories will still have linear pattern retrieval capacity even if the

patterns have good linear algebraic structures.

In [10], we presented some preliminary results in which two efficient recall algorithms were proposed for the case where the neural graph had the structure of an expander [11]. Here, we extend the previous results to general sparse neural graphs as well as proposing a simple learning algorithm to capture the internal structure of the patterns (which will be used later in the recall phase).

Due to their structure and capability to retrieve patterns from partially available information, associative memories have natural applications in content-addressable memories [12] as well as search engine algorithms that use not only user's inputs but also the association between the objects in the search domain (see [13] for example). Furthermore, they have also some strong links to modern error correcting codes [7] which use message passing over (bipartite) graphs to eliminate noise in communication channels. However, the inefficiency of current neural associative memories in reliably memorizing a large number of patterns acts as a barrier in deploying them in large-scale practical systems. We expect that improving the pattern retrieval capacity, even if it comes with some mild restrictions, will bring us one step closer to widespread adoption in practical systems.

The remainder of this paper is organized as follows: In Section II, we will discuss the neural model used in this paper and formally define the associative memory problem. We explain the proposed learning algorithm in Section III. Sections IV and V are respectively dedicated to the recall algorithm and analytically investigating its performance in retrieving corrupted patterns. In Section VI we address the pattern retrieval capacity and show that it is exponential in n. Simulation results are discussed in Section VII. Section VIII concludes the paper and discusses future research topics. Finally, the Appendix containsthe proofs for certain lemmas.

II. Problem Formulation and the Neural Model

A. The Model

In the proposed model, we work with neurons whose states are integers from a finite set of non-negative values $\mathcal{Q} = \{0, 1, \dots, Q-1\}$. A natural way of interpreting this model is to think of the integer states as the short-term firing rate of neurons (possibly quantized).

Like in other neural networks, neurons can only perform simple operations. We consider neurons that can do *linear summation* over the input and possibly apply a *non-linear function* (such as thresholding) to produce the output. More specifically, neuron x updates its state based on the states of its neighbors $\{s_i\}_{i=1}^{n}$ as follows:

1) It computes the weighted sum $h = \sum_{i=1}^{n} w_i s_i$, where w_i is the weight of the input link from the i^{th} neighbor.
2) It updates its state as $x = f(h)$, where $f : \mathbb{R} \to \mathcal{Q}$ is a possibly non-linear function.

We will refer to these two as "neural operations" in the sequel.

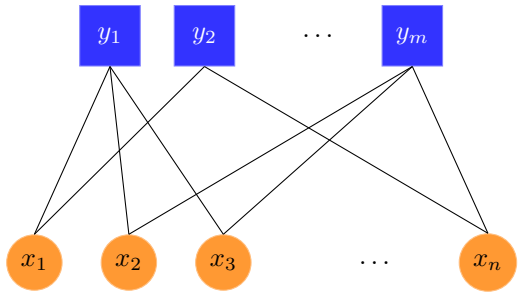

Fig. 1. A bipartite graph that represents the constraints on the training set. The weights are bidirectional. However, depending on the recall algorithm (which is explained later), the weight from y_i to x_j (W^b_{ij}) could be either equal to the weight from x_j to y_i (W_{ij}) or its sign. In other words, we have either $W^b_{ij} = \text{sign}(W_{ij})$ or $W^b_{ij} = W_{ij}$, depending on the algorithm used in the recall phase.

B. The Problem

The neural associative memory problem consists of two parts: learning and pattern retrieval.

1) The learning phase: We assume to be given C vectors of length n with integer-valued entries belonging to $\mathcal{Q}$. Furthermore, we assume these patterns belong to a subspace of $\mathcal{Q}^n$ with dimension $k < n$. Let $\mathcal{X}_{C\times n}$ be the matrix that contains the set of patterns in its rows. Note that if $k = n$ we are back to the original associative memory problem. Let us denote the model specification by a triplet $(\mathcal{Q}, n, k)$.

The learning phase then comprises a set of steps to determine the connectivity of the neural graph (i.e. finding a set of weights) as a function of the training patterns in $\mathcal{X}$ such that these patterns are stable states of the recall process. More specifically, in the learning phase we would like to memorize the patterns in $\mathcal{X}$ by finding a set of non-zero vectors $w_1, \dots, w_m \in \mathbb{R}^n$, with $m \leq n - k$, that are orthogonal to the set of given patterns. Note that such vectors are guaranteed to exist, one example being a basis for the null-space.

The inherent structure of the patterns are captured in the obtained null-space vectors, denoted by the matrix $W \in \mathbb{R}^{m\times n}$, whose i^{th} row is w_i. This matrix can be interpreted as the adjacency matrix of a bipartite graph which represents our neural network. The graph is comprised of pattern and constraint neurons (nodes). Pattern neurons, as their name suggests, correspond to the states of the patterns we would like to learn or recall. The constraint neurons, on the other hand, should verify if the current pattern belongs to the database $\mathcal{X}$. If not, they should send proper feedback messages to the pattern neurons in order to help them converge to the correct pattern in the dataset. The overall model is shown in Fig. 1.

2) The recall phase: In the recall phase, the neural network should retrieve the correct memorized pattern from a possibly corrupted version. In this case, the states of the pattern neurons $x_1, x_2, \dots, x_n$ are initialized with the given (noisy) input pattern. Here, we assume that the noise is integer valued

and additive[1]. Therefore, assuming the input to the network is a corrupted version of pattern x^μ, the state of the pattern nodes are $x = x^\mu + z$, where z is the noise. Now the neural network should use the given states together with the fact that $Wx^\mu = 0$ to retrieve pattern x^μ, i.e. it should estimate z from $Wx = Wz$ and return $x^\mu = x - z$. Any algorithm designed for this purpose should be simple enough to be implemented by neurons. Therefore, our objective is to find a simple algorithm capable of eliminating noise using only neural operations.

C. Related Work

Designing a neural associative memory has been an active area of research for the past three decades. Hopfield was among the first to design an artificial neural associative memory in his seminal work in 1982 [1]. The so-called Hopfield network is inspired by Hebbian learning [14] and is composed of binary-valued (± 1) neurons, which together are able to memorize a certain number of patterns. In our terminology, the Hopfield network corresponds to a $(\{-1, 1\}, n, n)$ neural model. The pattern retrieval capacity of a Hopfield network of n neurons was derived later by Amit et al. [15] and shown to be $0.13n$, under vanishing bit error probability requirement. Later, McEliece et al. [9] proved that under the requirement of vanishing pattern error probability, the capacity of Hopfield networks is $n/(2\log(n))) = O(n/\log(n))$.

In addition to neural networks with online learning capability, offline methods have also been used to design neural associative memories. For instance, in [2] the authors assume the complete set of pattern is given in advance and calculate the weight matrix using the pseudo-inverse rule [16] offline. In return, this approach helps them improve the capacity of a Hopfield network to $n/2$, under vanishing pattern error probability condition, while being able to correct *one bit* of error in the recall phase. Although this is a significant improvement, it comes at the price of much higher computational complexity and the lack of online learning ability.

While the connectivity graph of a Hopfield network is a complete graph, Komlos and Paturi [17] extended the work of McEliece to sparse neural graphs. Their results are of particular interest as physiological data is also in favor of sparsely interconnected neural networks. They have considered a network in which each neuron is connected to d other neurons, i.e., a d-regular network. Assuming that the network graph satisfies certain connectivity measures, they prove that it is possible to store $C = O(d/\log n)$ *random* patterns with vanishing pattern error probability. Furthermore, they show that in spite of the capacity reduction, the error correction capability remains the same as the network can still tolerate a number of errors which is linear in n.

It is also known that the capacity of neural associative memories could be enhanced if the patterns are of *low-activity* nature, in the sense that at any moment many of the neurons are silent [16]. However, even these schemes fail when required to correct a fair amount of erroneous bits as the information retrieval is not better than that of normal networks.

Extension of associative memories to non-binary neural models has also been explored in the past. Hopfield addressed the case of continuous neurons and showed that similar to the binary case, neurons with states between -1 and 1 can memorize a set of random patterns, albeit with less capacity [18]. In [3] the authors investigated a multi-state complex-valued neural associative memory for which the estimated capacity is $C < 0.15n$. Under the same model but using a different learning method, Muezzinoglu et al. [4] showed that the capacity can be increased to $C = n$. However the complexity of the weight computation mechanism is prohibitive. To overcome this drawback, a Modified Gradient Descent learning Rule (MGDR) was devised in [19]. In our terminology, all of these models are $(\{e^{2\pi j s/k} | 0 \leq s \leq k-1\}, n, n)$ neural associative memories.

Given that even very complex offline learning methods can not improve the capacity of binary or multi-state neural associative memories, a group of recent works has made considerable efforts to exploit the inherent structure of the patterns in order to increase capacity and improve error correction capabilities. Such methods focus merely on memorizing those patterns that have some sort of inherent redundancy. As a result, they differ from previous methods in which the network was designed to be able to memorize any random set of patterns. Pioneering this approach, Berrou and Gripon [20] achieved considerable improvements in the pattern retrieval capacity of Hopfield networks, by utilizing Walsh-Hadamard sequences. Walsh-Hadamard sequences are a particular type of low correlation sequences and were initially used in CDMA communications to overcome the effect of noise. The only slight downside to the proposed method is the use of a decoder based on the winner-take-all approach which requires a separate neural stage, increasing the complexity of the overall method. Using low correlation sequences has also been considered in [5], where the authors introduced two novel mechanisms of neural association that employ binary neurons to memorize patterns belonging to another type of low correlation sequences, called Gold family [21]. The network itself is very similar to that of Hopfield. However, the authors failed to increase the pattern retrieval capacity beyond $C = n$.

Later, Gripon and Berrou came up with a different approach based on neural cliques, which increased the pattern retrieval capacity to $O(n^2)$ [6]. Their method is based on dividing a neural network of size n into c clusters of size n/c each. Then, the messages are chosen such that only one neuron in each cluster is active for a given message. Therefore, one can think of messages as a random vector of length $c\log(n/c)$, where the $\log(n/c)$ part specifies the index of the active neuron in a given cluster. The authors also provide a learning algorithm, similar to that of Hopfield, to learn the pairwise correlations within the patterns. Using this technique and exploiting the fact that the resulting patterns are very sparse, they could boost the capacity to $O(n^2)$ while maintaining the computational

[1]It must be mentioned that neural states below 0 and above $Q-1$ will be clipped to 0 and $Q-1$, respectively. This is biologically justified as the firing rate of neurons can not exceed an upper bound and of course can not be less than zero.

simplicity of Hopfield networks.

Modification of neural architecture has been also used in [22] to increased the capacity. Here, the network is divided into b smaller fully interconnected *disjoint* blocks of size n/b. Using this approach, the capacity is increased to $\Theta\left(b^{n/b}\right)$ (for *random* patterns), with $b = \omega(\ln n)$. This is a huge improvement but comes at the price of limited *worst-case* noise tolerance capabilities. More specifically, since the network is a set of disjoint Hopfield networks of size b, the amount of error each block could correct is in the order of ϵb, for some constant $\epsilon > 0$. As a result, in worst-case scenarios where the error is not spread uniformly over the network and is concentrated on some clusters, the error correction is limited by the performance of individual blocks.

In contrast to the pairwise correlation of the Hopfield model, Peretto et al. [23] deployed *higher order* neural models: the models in which the state of the neurons not only depends on the state of their neighbors, but also on the correlation among them. Under this model, they showed that the storage capacity of a higher-order Hopfield network can be improved to $C = O(n^{p-2})$, where p is the degree of correlation considered. The main drawback of this approach is the huge computational complexity required in the learning phase, as one has to keep track of $O(n^{p-2})$ neural links and their weights during the learning period.

Recently, the present authors introduced a novel model inspired by modern coding techniques in which a neural bipartite graph is used to memorize the patterns that belong to a subspace [10]. The proposed model can be also thought of as a way to capture higher order correlations in given patterns while keeping the computational complexity to a minimal level (since instead of $O(n^{p-2})$ weights one needs to only keep track of $O(n^2)$ of them). Under the assumptions that the bipartite graph is known, sparse, and expander, the proposed algorithm increased the pattern retrieval capacity to $C = O(a^n)$, for some $a > 1$. The main drawbacks in the proposed approach were the lack of a learning algorithm as well as the expansion assumption on the neural graph.

Although the model proposed in [10] is based on bipartite graphs, it performs auto-association and, thus, differs from (hetero-associative) Bidirectional Associative Memory [24] and other associative memories performing Self-Organizing Maps [25] for the purpose of classification or clustering [26], [27]. Furthermore, the model in [10] also differs from feed-forward associative memories in the sense that it uses back and forward message passing to perform the recall task.

In this paper, we focus on extending the results of [10] in several directions: first, we suggest an iterative learning algorithm to find the neural connectivity matrix from the patterns in the training set. Secondly, we provide an analysis of the proposed error correcting algorithm in the recall phase and investigate its performance for different network models. We also show that our proposed algorithm is capable of memorizing an exponential number of patterns Finally, we discuss some variants of the error correcting method which achieve better performance in practice.

It is worth mentioning that an extension of this approach to a multi-level neural network is considered in [28]. There, the novel structure enables better error correction. However, the learning algorithm lacks the ability to learn the patterns one by one and requires the patterns to be presented all at the same time in the form of a big matrix. In [29] we have further extended this approach to a modular single-layer architecture with online learning capabilities, which is very similar to the learning algorithm we are going to explain later. The modular structure makes the recall algorithm much more efficient.

Another important point to note is that learning linear constraints by a neural network is hardly a new topic as one can learn a matrix orthogonal to the patterns in the training set (i.e. $Wx^\mu = 0$) using simple neural learning rules (we refer the interested readers to [30] and [31]). However, to the best of our knowledge, finding such a matrix subject to the sparsity constraints has not been investigated before. This problem can also be regarded as an instance of compressed sensing [32], in which the measurement matrix is the big patterns matrix $\mathcal{X}_{C\times n}$ and the set of measurements are the orthogonality constraints. Thus, we are interested in finding a sparse vector w such that $\mathcal{X}w = 0$. Nevertheless, many decoders proposed in this area are very complex and cannot be implemented by a neural network using simple neuron operations. Some exceptions are [33] and [34] which are closely related to the learning algorithm proposed in this paper.

D. Solution Overview

Before going through the details of the algorithms, let us give an overview of the proposed solution. To learn the set of given patterns, we have adopted the neural learning algorithm proposed in [35] and modified it to favor sparse solutions. In each iteration of the algorithm, a random pattern from the data set is picked and the neural weights corresponding to constraint neurons are adjusted is such a way that the projection of the pattern along the current weight vectors is reduced, while trying to make the weights sparse as well.

In the recall phase, we exploit the fact that the learned neural graph is sparse and orthogonal to the set of patterns. Therefore, when a query is given, if it is not orthogonal to the connectivity matrix of the weighted neural graph, it must have been corrupted by noise. We will use the sparsity of the neural graph to eliminate this noise using a simple iterative algorithm. In each iteration, there is a set of violated constraint neurons, i.e. those that receive a non-zero weighted sum over their input links. These nodes will send feedback to the pattern neurons that they are connected to (i.e. their neighbors), where the feedback is the sign of the received input-sum. At this point, the pattern nodes that receive feedback from a majority of their neighbors update their state according to the sign of the sum of received messages. This process continues until noise is eliminated completely or a failure is declared.

III. Learning Phase

Since the patterns are assumed to be coming from a subspace in the n-dimensional space, we adapt the algorithm

proposed by Oja and Karhunen [35] to learn the null-space basis of the subspace defined by the patterns. In fact, a very similar algorithm is also used in [30] for the same purpose. However, since we need the basis vectors to be sparse (due to requirements of the recall algorithms), we add an additional term to penalize non-sparse solutions during the learning.

Another difference with the proposed method and that of [30] is that the learning algorithm proposed in [30] yields dual vectors that form an orthogonal set. Although one can easily extend our suggested method to such a case as well, we find this requirement unnecessary in our case. This gives us the additional advantage to make the algorithm *parallel* and *adaptive*. Parallel in the sense that we can repeat the same algortihm separately several times to find all constraints with high probability. And adaptive in the sense that we can determine the number of constraints on-the-go, i.e. start by learning just a few constraints. If needed (for instance due to bad performance in the recall phase), the network can easily learn additional constraints. This increases the flexibility of the algorithm and provides a nice trade-off between the time spent on learning and the performance in the recall phase. Both these points make an approach biologically realistic.

It should be mentioned that the core of our learning algorithm is virtually the same as the one we proposed in [29].

A. Overview of the proposed algorithm

The problem to find one sparse constraint *vector* w is given by equations (1a), (1b), in which pattern μ is denoted by x^μ.

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

subject to:

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

In the above problem, $\cdot$ is the inner-product, $\|.\|_2$ represent the ℓ_2 vector norm, $g(w)$ a penalty function to encourage sparsity and η is a positive constant. There are various ways to choose $g(w)$. For instance one can pick $g(w)$ to be $\|.\|_1$, which leads to ℓ_1-norm penalty and is widely used in compressed sensing applications [33], [34]. Here, we will use a different penalty function, as explained later.

To form the basis for the null space of the patterns, we need $m = n-k$ vectors, which we can obtain by solving the above problem several times, each time from a random initial point[2].

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

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

where σ is a constant that should be chosen appropriately. Intuitively, $\tanh(\sigma w_i^2)$ approximates $|\text{sign}(w_i)|$ in ℓ_0-norm. Therefore, the larger σ is, the closer $g(w)$ will be to $\|.\|_0$.

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

By calculating the derivative of the objective function, and by considering the update due to each randomly picked pattern x, we will get the following iterative algorithm:

$$y(t) = x(t) \cdot w(t) \tag{2a}$$

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

$$w(t+1) = \frac{\tilde{w}(t+1)}{\|\tilde{w}(t+1)\|_2} \tag{2c}$$

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

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

It is easy to see that $\Gamma_i(w(t)) \simeq 2\sigma w_i(t)$ for relatively small $w_i(t)$'s. And for larger values of $w_i(t)$, we get $\Gamma_i(w(t)) \simeq 0$. Therefore, by proper choice of η and σ, equation (2b) suppresses small entries of $w(t)$ by pushing them towards zero, thus, favoring sparser results. To simplify the analysis, and with some abuse of notation, we approximate the function $\Gamma(w(t))$ with the following function:

$$\Gamma_i(w(t)) = \begin{cases} w_i(t) & \text{if } |w_i(t)| \le \theta_t; \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where θ_t is a small positive threshold.

Following the same approach as [35] and assuming α_t to be small enough such that equation (2c) can be expanded as powers of α_t, we can approximate equation (2) with the following simpler version:

$$y(t) = x(t) \cdot w(t) \tag{4a}$$

$$w(t+1) = w(t) - \alpha_t \left(y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right) + \eta\Gamma(w(t)) \right) \tag{4b}$$

In the above approximation, we also omitted the term $\alpha_t\eta\left(w(t)\cdot\Gamma(w(t))\right)w(t)$ since $w(t)\cdot\Gamma(w(t))$ would be negligible, specially as θ_t in equation (3) becomes smaller.

The overall learning algorithm for one constraint node is given by Algorithm 1. In words, in Algorithm 1 $y(t)$ is the projection of $x(t)$ on the basis vector $w(t)$. If for a given data vector $x(t)$, $y(t)$ is equal to zero, namely, the data is orthogonal to the current weight vector $w(t)$, then according to equation (4b) the weight vector will not be updated. However, if $x(t)$ has some projection over $w(t)$ then the weights are updated towards the direction to reduce this projection.

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

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

Algorithm 1 Iterative Learning

Input: Set of patterns $x^\mu \in \mathcal{X}$ with $\mu = 1, \ldots, C$, stopping point ε.

Output: w

while $\sum_\mu |x^\mu \cdot w(t)|^2 > \varepsilon$ **do**

Choose $x(t)$ at random from patterns in $\mathcal{X}$

Compute $y(t) = x(t) \cdot w(t)$

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

$t \leftarrow t + 1$.

end while

Remark 1. *Although we are interested in finding a sparse graph, note that too much sparseness is not desired. This is because we are going to use the feedback sent by the constraint nodes to eliminate input noise during the recall phase. If the graph is too sparse, the number of feedback messages received by each pattern node is too small to be relied upon. Therefore, we must adjust the penalty coefficient η properly to get a* sufficiently *sparse neural graph.*

B. Convergence analysis

To prove the convergence of Algorithm 1, let $A = \mathbb{E}\{xx^T | x \in \mathcal{X}\}$ be the correlation matrix for the patterns in the training set. Also, denote $A_t = x(t)(x(t))^\top$, (hence, $A = \mathbb{E}(A_t)$). Furthermore, let $E(t) = E(w(t)) = \frac{1}{C}\sum_{\mu=1}^{C} (w(t)^\top x^\mu)^2$ be the objective function. Finally, assume that the learning rate α_t is small enough so that terms that are $O(\alpha_t^2)$can be neglected, similar to approximation we made in deriving equation (4). In general, we pick the learning rate $\alpha_t \propto 1/t$ so that $\alpha_t > 0$, $\sum \alpha_t \to \infty$ and $\sum \alpha_t^2 < \infty$. We first show that the weight vector $w(t)$ never becomes zero, i.e. $\|w(t)\|_2 > 0$ for all t.

Lemma 1. *Assume we initialize the weights such that $\|w(0)\|_2 > 0$. Furthermore, assume $\alpha_t < \alpha_0 < 1/(2\eta)$. Then, for all iterations t we have $\|w(t)\|_2 > 0$.*

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

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

Thus, in order to have $\|w(t+1)\|_2 > 0$, we must have that $\|w'(t)\|_2 - 2\alpha_t \eta \|\Gamma(w(t))\|_2 > 0$. Given that, $\|\Gamma(w(t))\|_2 \leq \|w(t)\|_2 \leq \|w'(t)\|_2$, it is sufficient to have $2\alpha_t \eta < 1$ in order to achieve the desired inequality. This proves the lemma. ∎

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

Theorem 2. *Suppose the learning rate α_t is sufficiently small and both the learning rate α_t and the sparsity threshold θ_t decay according to $1/t$. Then, Algorithm 1 converges to a local minimum w^* for which $E(w^*) = 0$. At this point, w^* is orthogonal to the patterns in the data set $\mathcal{X}$.*

Proof: From equation (4b) recall that

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

Let $Y(t) = \mathbb{E}_x(\mathcal{X} w(t))$. Thus, we will have

$$\begin{aligned} Y(t+1) &= Y(t) \left(1 + \alpha_t \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} \right) \\ &- \alpha_t \left(\mathcal{X} A w(t) + \eta \mathcal{X} \Gamma(w(t)) \right) \end{aligned}$$

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

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

In the above equations, approximation (a) is obtained by omitting all terms that are $O(\alpha_t)^2$. Approximation (b) follows by noting that $\alpha_t \eta \|(w(t))^\top A \Gamma(w(t))\|_2 \leq \alpha_t \eta \|w(t)\|_2 \|A\|_2 \|\Gamma(w(t))\|_2 \leq \alpha_t \eta \|w(t)\|_2 \|A\|_2 (n\theta_t)$. Now since $\theta_t = \Theta(\alpha_t)$, $\alpha_t \eta \|(w(t))^\top A \Gamma(w(t))\|_2 = O(\alpha_t^2)$ and, hence, can be eliminated as well.

Therefore, in order to show that the algorithm converges, we need to show that $\left(w(t)^\top A^2 w(t) - \frac{w(t)^\top A w(t)}{\|w(t)\|_2^2} E(t) \right) \geq 0$ to have $E(t+1) \leq E(t)$. Noting that $E(t) = w(t)^\top A w(t)$, we must show that $w^\top A^2 w \geq (w^\top A w)^2 / \|w\|_2^2$. Note that the left hand side is $\|Aw\|_2^2$. For the right hand side, we have

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

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

Remark 2. *Note that the above theorem only proves that the obtained vector is orthogonal to the dataset and says nothing about its degree of sparsity. The reason is that there is no guarantee that the dual basis of a subspace be sparse. However, our experimental results in section VII show that the introduction of a sparsity penalty function ($\Gamma(w)$) works perfectly well and the learning algorithm results in sparse solutions.*

C. Running the Algorithm in Parallel

In order to find m constraints, we need to repeat Algorithm 1 several times. Fortunately, we can repeat this process in parallel, which speeds up the algorithm and is more meaningful from a biological point of view as each constraint neuron can act independently of other neighbors. Although doing the algorithm in parallel may result in linearly dependent constraints once in a while, our experimental results show that starting from different random initial points, the algorithm converges to different distinct constraints most of the time.

IV. Recall Phase

In the recall phase, we are going to exploit the fact that our learning algorithm has resulted in the connectivity matrix of a neural graph which is sparse and orthogonal to the memorized patterns. Therefore, given a noisy version of the learned patterns, we can use the feedback from the constraint neurons in Fig. 1 to design an algorithm which eliminates noise. More specifically, the linear input sums to the constraint neurons are given by the elements of the vector $W(x^\mu + z) = Wx^\mu + Wz = Wz$, with z being the integer-valued input noise (biologically speaking, the noise can be interpreted as a neuron skipping some spikes or firing more spikes than it should). Based on observing the elements of Wz, each constraint neuron feeds back a message (containing info about z) to its neighboring pattern neurons. Based on this feedback, and exploiting the fact that W is sparse, the pattern neurons update their states in order to reduce the noise z.

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

A. The Recall Algorithms

The proposed algorithm for the recall phase comprises a series of forward and backward iterations. Two different methods are suggested in this paper, which slightly differ from each other in the way pattern neurons are updated. The first one is based on the Winner-Take-All approach (WTA) and is given by Algorithm 2. In this version, only the pattern node that receives the highest amount of normalized feedback updates its state while the other pattern neurons maintain their current states. The normalization is done with respect to the degree of each pattern neuron, i.e. the number of edges connected to each pattern neuron in the neural graph. The WTA circuitry can be easily added to the neural model shown in Fig. 1 using any of the classic WTA methods [16].

Algorithm 2 Recall Algorithm: Winner-Take-All

Input: Connectivity matrix W, iteration $t_{\max}$
Output: $x_1, x_2, \ldots, x_n$

1: **for** $t = 1 \to t_{\max}$ **do**
2: *Forward iteration:* Calculate $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each constraint neuron and set $y_i = \text{sign}(h_i)$.
3: *Backward iteration:* Each neuron x_j with degree d_j computes

$$g_j^{(1)} = \frac{\sum_{i=1}^{m} W^b_{ij} y_i}{d_j}, g_j^{(2)} = \frac{\sum_{i=1}^{m} |W^b_{ij} y_i|}{d_j}$$

4: Find

$$j^* = \arg\max_j g_j^{(2)}.$$

5: Update the state of winner: set $x_{j^*} = x_{j^*} - \text{sign}(g_{j^*}^{(1)})$.
6: **end for**

Algorithm 3 Recall Algorithm: Majority-Voting

Input: Connectivity matrix W, threshold φ, iteration $t_{\max}$
Output: $x_1, x_2, \ldots, x_n$

1: **for** $t = 1 \to t_{\max}$ **do**
2: *Forward iteration:* Calculate $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each constraint neuron and set $y_i = \text{sign}(h_i)$.
3: *Backward iteration:* Each neuron x_j with degree d_j computes

$$g_j^{(1)} = \frac{\sum_{i=1}^{m} W^b_{ij} y_i}{d_j}, g_j^{(2)} = \frac{\sum_{i=1}^{m} |W^b_{ij} y_i|}{d_j}$$

4: Update the state of each pattern neuron j according to $x_j = x_j - \text{sign}(g_j^1)$ only if $|g_j^{(2)}| > \varphi$.
5: **end for**

The second approach, given by Algorithm 3, is much simpler: in every iteration, each pattern neuron decides locally whether or not to update its current state. More specifically, if the amount of feedback received by a pattern neuron exceeds a threshold, the neuron updates its state; otherwise, it remains unchanged.[4] In both algorithms, the quantity $g_j^{(2)}$ can be interpreted as the number of feedback messages received by pattern neuron x_j from the constraint neurons. On the other hand, the sign of $g_j^{(1)}$ provides an indication of the sign of the

[4]Note that in order to maintain the current value of a neuron in case no input feedback is received, we can add self-loops to pattern neurons in Fig. 1. These self-loops are not shown in the figure for clarity.

noise that affects x_j, and $|g_j^{(1)}|$ indicates the confidence level in the decision regarding the sign of the noise.

It is worthwhile mentioning that the Majority-Voting decoding algorithm is very similar to the Bit-Flipping algorithm of Sipser and Spielman to decode LDPC codes [37] and a similar approach in [38] for compressive sensing methods.

Remark 3. *To give the reader some insight about why the neural graph should be sparse in order for the above algorithms to work, consider the backward iteration of both algorithms: it is based on counting the fraction of received input feedback messages from the neighbors of a pattern neuron. In the extreme case, if the neural graph is complete, then a single noisy pattern neuron results in the violation of all constraint neurons in the forward iteration. Consequently, in the backward iteration all pattern neurons receive feedback from their neighbors and it is impossible to tell which of them is the noisy one.*

However, if the graph is sparse, a single noisy pattern neuron only makes some of the constraints unsatisfied. As a result, in the backward iteration only the nodes which share the neighborhood of the noisy node receive some feedback. And the fraction of the received feedback messages would be much larger for the original noisy node. Therefore, by merely looking at this fraction, one can identify the noisy pattern neuron with high probability as long as the graph is sparse and the input noise is reasonably bounded.

B. Some Practical Modifications

Although Algorithm 3 is fairly simple and practical, each pattern neuron still needs two types of information: the number of received feedbacks and the net input sum. We can modify the recall algorithm to make it more practical and simpler by replacing the term $\|w_j\|_0 = d_j$ with $\|w_j\|_1$. Furthermore, we assume symmetric weights, i.e $W_{ij}^b = W_{ij}$.

Interestingly, in some of our experimental results corresponding to *denser graphs*, this approach performs better, as will be illustrated in section VII. One possible reason behind this improvement might be the fact that using the ℓ_1-norm instead of the ℓ_0-norm will result in better differentiation between two vectors that have the same number of non-zero elements but differ in magnitudes of the elements.

V. Performance Analysis

In order to obtain analytical estimates on the recall probability of error, we assume that the connectivity graph W is sparse. With respect to this graph, we define the pattern and constraint degree distributions as follows.

Definition 1. *For the bipartite graph W, let λ_i (ρ_j) denote the fraction of edges that are adjacent to pattern (constraint) nodes of degree i (j). We call $\{\lambda_1, \ldots, \lambda_m\}$ and $\{\rho_1, \ldots, \rho_n\}$ the pattern and constraint degree distribution form the edge perspective, respectively. Furthermore, it is convenient to define the degree distribution polynomials as*

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

The degree distributions are determined after the learning phase is finished and in this section we assume they are given. Furthermore, we consider an ensemble of random neural graphs with a given degree distribution and investigate the average performance of the recall algorithms over this ensemble. Here, the word "ensemble" refers to the assumption of having a number of *random* neural graphs with the given degree distributions and do the analysis for the average scenario.

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

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

We do the analysis only for the Majority-Voting algorithms since if we choose the Majority-Voting update threshold $\varphi = 1$, roughly speaking, we will have the WTA algorithm.[5]

As mentioned earlier, in this paper we will perform the analysis for general sparse bipartite graphs. However, restricting ourselves to a particular type of sparse graphs known as "expander" allows us to prove stronger results on the recall error probabilities. More details can be found in [39] and in [10]. However, since it is very difficult, if not impossible in certain cases, to make a graph expander during an iterative learning method, we focus on the more general case of sparse neural graphs.

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

Based on the error correcting algorithm and the above notations, in a given iteration two types of errors are possible:

1) Type-1 errors: A node $x \in \mathcal{C}_t$ decides to update its value. The probability of this event is denoted by $P_{e_1}(t)$.
2) Type-2 errors: A node $x \in \mathcal{E}_t$ updates its value in the wrong direction. Let $P_{e_2}(t)$ denote the probability of error for this type.

We start the analysis by finding explicit expressions and upper bounds on the average of $P_{e_1}(t)$ and $P_{e_2}(t)$ over all nodes as a function S_t. We then find an exact relationship for S_t as a function of $|\mathcal{E}_t|$, which will provide us with the

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

required expressions on the average bit error probability as a function of the number of noisy input symbols, $|\mathcal{E}_0|$. Having found the average bit error probability, we can easily bound the block error probability for the recall algorithm.

A. Error probability - type 1

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

$$P_1^x(t) = \Pr\{\frac{|\mathcal{N}(\mathcal{E}_t) \cap \mathcal{N}(x)|}{d_x} \geq \varphi\} \tag{5}$$

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

$$P_1^x(t) \approx \sum_{i=\lceil \varphi d_x \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t}{m}\right)^i \left(1 - \frac{S_t}{m}\right)^{d_x - i}. \tag{6}$$

In the above equation, S_t/m represents the probability of having one of the d_x edges connected to the S_t constraint neurons that are neighbors of the erroneous pattern neurons.

As a result of the above equations, we have:

$$P_{e_1}(t) = \mathbb{E}_{d_x}(P_1^x(t)), \tag{7}$$

where $\mathbb{E}_{d_x}$ denote the expectation over the degree distribution $\{\lambda_1, \ldots, \lambda_m\}$.

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

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

B. Error probability - type 2

A node $x \in \mathcal{E}_t$ makes a wrong decision if the net input sum it receives has a different sign than the sign of noise it experiences. Instead of finding an exact relation, we bound this probability by the probability that the neuron x shares at least half of its neighbors with other neurons, i.e. $P_{e_2}(t) \leq \Pr\{\frac{|\mathcal{N}(\mathcal{E^*}_t) \cap \mathcal{N}(x)|}{d_x} \geq 1/2\}$, where $\mathcal{E^*}_t = \mathcal{E}_t \setminus x$. Letting $P_2^x(t) = \Pr\{\frac{|\mathcal{N}(\mathcal{E^*}_t) \cap \mathcal{N}(x)|}{d_x} \geq 1/2 | \deg(x) = d_x\}$, we will have:

$$P_2^x(t) = \sum_{i=\lceil d_x/2 \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t^*}{m}\right)^i \left(1 - \frac{S_t^*}{m}\right)^{d_x - i} \tag{8}$$

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

Therefore, we will have:

$$P_{e_2}(t) \leq \mathbb{E}_{d_x}(P_2^x(t)) \tag{9}$$

Combining equations (7) and (9), the bit error probability at iteration t would be

$$\begin{aligned} P_b(t+1) &= \Pr\{x \in \mathcal{C}_t\} P_{e_1}(t) + \Pr\{x \in \mathcal{E}_t\} P_{e_2}(t) \\ &= \frac{n - |\mathcal{E}_t|}{n} P_{e_1}(t) + \frac{|\mathcal{E}_t|}{n} P_{e_2}(t) \end{aligned} \tag{10}$$

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

$$P_e(t) = 1 - (1 - P_b(t))^n \tag{11}$$

Equation (11) gives the probability of making a mistake in iteration t. Therefore, we can bound the overall probability of error, P_E, by setting $P_E = \lim_{t \to \infty} P_e(t)$. To this end, we have to recursively update $P_b(t)$ in equation (10) and using $|\mathcal{E}_{t+1}| \approx n P_b(t+1)$. However, since we have assumed that the noise values are ± 1, we can provide an upper bound on the total probability of error by considering

$$P_E \quad \leq \quad P_e(1) \tag{12}$$

In other words, we assume that the recall algorithms either eliminate the input noise in the first iteration or a recall error is declared. Obviously, this bound is not as tight as possible and one might be able to correct errors in later iterations. In fact simulation results confirm this expectation. However, this approach provides a nice analytical upper bound since it only depends on the initial number of noisy nodes. As the initial number of noisy nodes grow, the above bound becomes tight. Thus, in summary we have

$$P_E \leq 1 - (1 - \frac{n - |\mathcal{E}_0|}{n} \bar{P}_1^x - \frac{|\mathcal{E}_0|}{n} \bar{P}_2^x)^n \tag{13}$$

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

Now, what remains to do is to find an expression for S_t and S_t^* as a function of $|\mathcal{E}_t|$. The following lemma will provide us with the required relationship.

Lemma 3. *The average neighborhood size S_t in iteration t is given by:*

$$S_t = m\left(1 - (1 - \frac{\bar{d}}{m})^{|\mathcal{E}_t|}\right) \tag{14}$$

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

Proof: The proof is given in Appendix A. ∎

To obtain S_t^* we could apply the above lemma with $|\mathcal{E}_t| - 1$ as the exponent in the right-hand side expression.

VI. Pattern Retrieval Capacity

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

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

Theorem 4. *Let $\mathcal{X}$ be a $C \times n$ matrix, formed by C vectors of length n with non-negative integers entries between 0 and $Q - 1$. Furthermore, let $k = rn$ for some $0 < r < 1$. Then,*

there exists a set of such vectors for which $C = a^{rn}$*, with* $a > 1$*, and rank*$(\mathcal{X}) = k < n$*.*

Proof: The proof is based on construction: we construct a data set $\mathcal{X}$ with the required properties. To start, consider a matrix $G \in \mathbb{R}^{k\times n}$ with rank k and $k = rn$, with $0 < r < 1$. Let the entries of G be non-negative integers, between 0 and $\gamma - 1$, with $\gamma \geq 2$.

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

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

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

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

$$Q - 1 \geq d^*(\gamma - 1)(\upsilon - 1) \tag{15}$$

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

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

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

VII. Simulation Results

A. Simulation Scenario

We have simulated the proposed learning and recall algorithms for three different network sizes $n = 200, 400, 800$, with $k = n/2$ for all cases. For each case, we considered a few different setups with different values for α, η, and θ in the learning algorithm 1, and different φ for the Majority-Voting recall algorithm 3. For brevity, we do not report all the results for various combinations but present only a selection of them to give insight on the performance of the proposed algorithms.

In all cases, we generated 50 random training sets using the approach explained in the proof of theorem 4, i.e. we generated a generator matrix G at random with $0/1$ entries and $d^* = 10$. We also used $0/1$ generating message words u and put $Q = 11$ to ensure the validity of the generated training set.

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

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

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

B. Learning Phase Results

In the learning algorithm, we go over the patterns in the dataset several times to make sure that update for one pattern does not adversely affect the other learned patterns. Let t be the number of times we have gone over the training set so far. Then we set $\alpha_t \propto \alpha_0/t$ to ensure the conditions of Theorem 2 is satisfied. Interestingly, all of the constraints converged in at most two learning iterations for all different setups. Therefore, the learning is very fast in this case.

Fig. 2 illustrates the percentage of pattern nodes with the specified sparsity degree defined as $\varrho = \kappa/n$, where κ is the number of non-zero elements. From the figure we notice two trends: Increasing the sparsity threshold θ_0 makes the network sparser and larger networks become sparser in general.

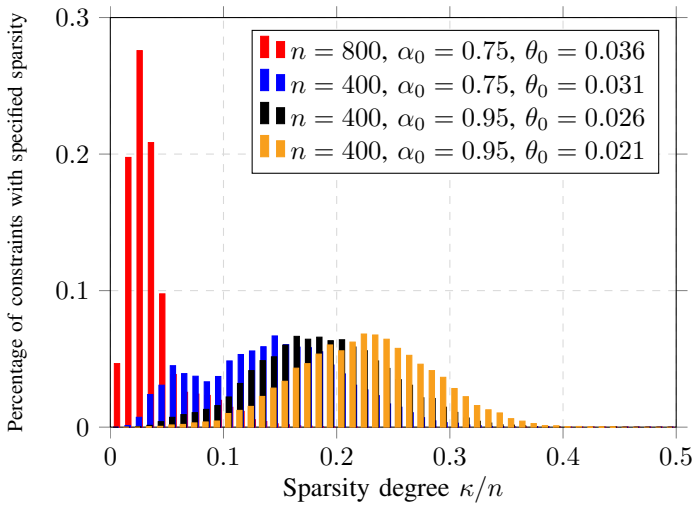

Fig. 2. The percentage of variable nodes with the specified sparsity degree and different values of network sizes and sparsity thresholds. The sparsity degree is defined as $\varrho = \kappa/n$, where κ is the number of non-zero elements.

C. Recall Phase Results

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

TABLE I
SIMULATION PARAMETERS

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

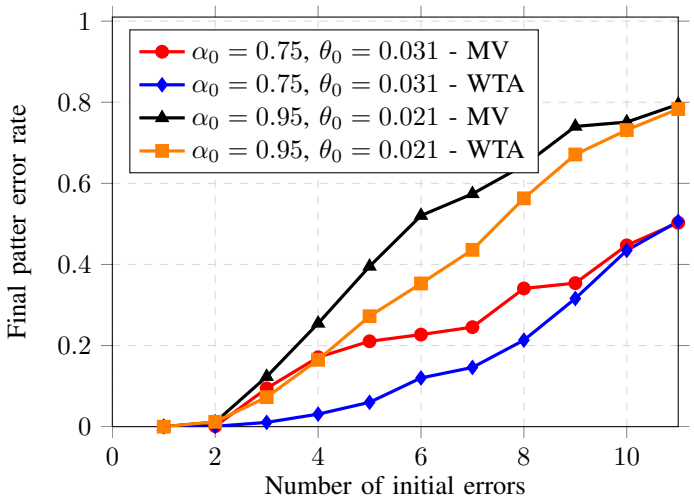

Fig. 3. Pattern error rate against the initial number of erroneous nodes for two different values of θ_0. Here, the network size is $n = 400$ and $k = 200$. The blue curves correspond to the sparser network (larger θ_0) and clearly shows a better performance.

Fig. 3 illustrates the effect of the sparsity threshold θ on the performance of the recall algorithm. Here, we have $n = 400$ and $k = 200$. Two different sparsity thresholds are compared together, namely $\theta_t \propto 0.031/t$ and $\theta_t \propto 0.021/t$. Clearly, as network becomes sparser, i.e. θ increases, the performance of both recall algorithms improve.

Fig. 4 illustrates the effect of network size on the performance of recall algorithms. As obvious from the figure, the performance improves to a great extent when we have a larger network. This is partially because of the fact that in larger networks, the connections are relatively sparser.

Fig. 5 compares the results obtained in simulation with the upper bound derived in Section V. Note that as expected, the bound is quite loose since in deriving inequality (11) we only considered the first iteration of the algorithm.

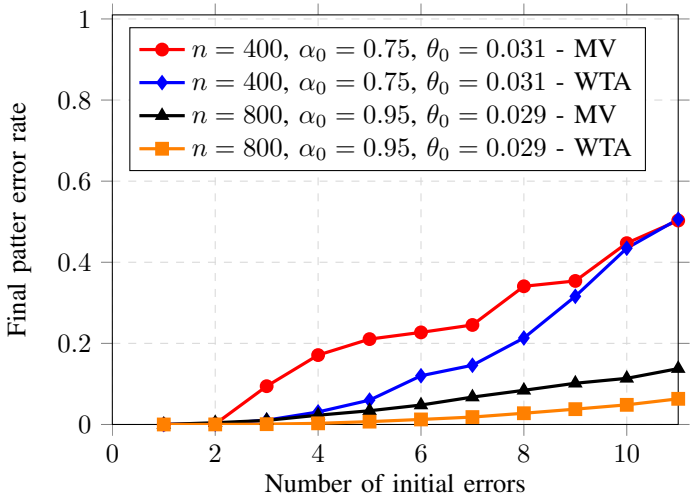

Fig. 4. Pattern error rate against the initial number of erroneous nodes for two different network sizes $n = 800$ and $k = 400$. In both cases $k = n/2$.

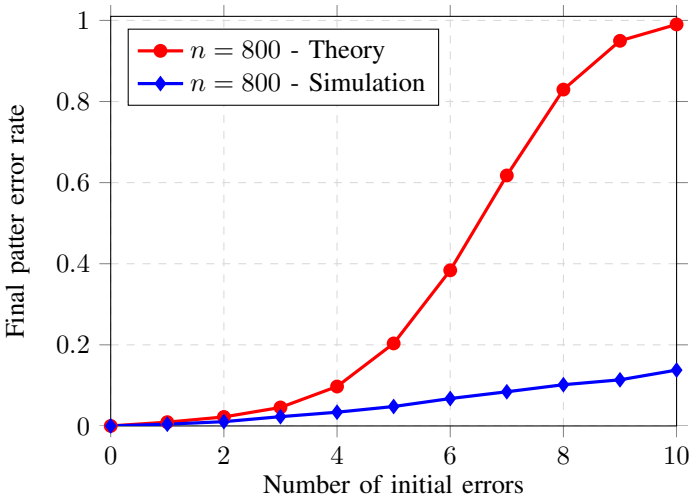

Fig. 5. Pattern error rate against the initial number of erroneous nodes and comparison with theoretical upper bounds for $n = 800$, $k = 400$, $\alpha_0 = 0.95$ and $\theta_0 = 0.029$.

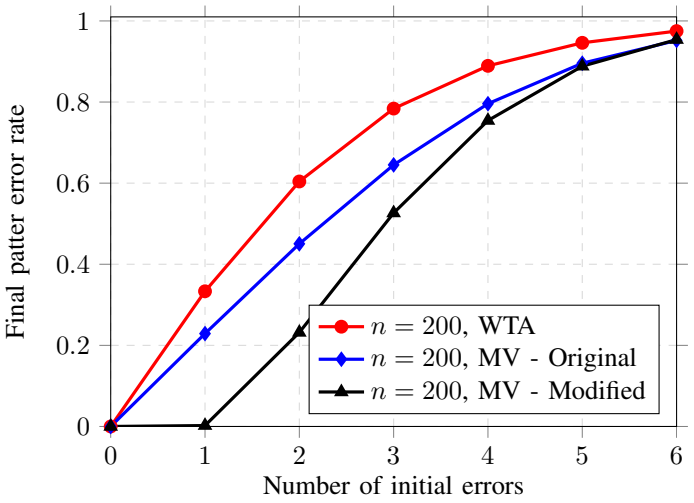

Fig. 6. Pattern error rate against the initial number of erroneous nodes for two different values of θ_0. Here, the network size is $n = 400$ and $k = 200$. The blue curves correspond to the sparser network (larger θ_0) and clearly show a better performance.

We also investigate the performance of the modified and more practical version of the Majority-Voting algorithm, which was explained in Section IV-B. Fig. 6 compares the performance of the WTA and original MV algorithms with the modified version of MV algorithm for a network with size $n = 200$, $k = 100$ and learning parameters $\alpha_t \propto 0.45/t$, $\eta = 0.45$ and $\theta_t \propto 0.015/t$. The neural graph of this particular example is rather dense, because of small n and θ. Here, the modified MV algorithm performs better because of the extra information provided by the ℓ_1-norm (compared to the ℓ_0-norm in the original MV algorithm). However, note that we did not observe this trend for the other simulation scenarios where the neural graph was sparser.

Finally, in Fig. 7 we compare perfromance of the proposed method in this paper and the multi-state complex-valued neural networks in [3] and [19]. To have a fair comparison, we first applied the methods of [3] and [19] to a dataset of randomly

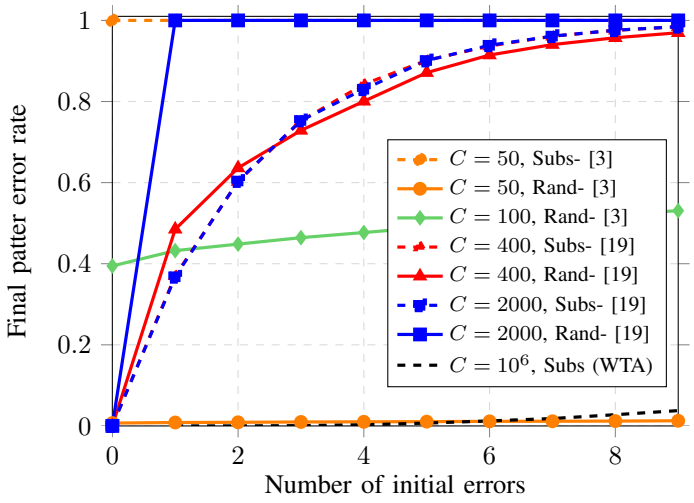

Fig. 7. Recall error rate against the initial number of errors for the method proposed in this paper and those of [3] and [19]. Different pattern retrieval capacities are considered to investigate their effect on the recall error rate.

generated patterns, which is the setting that these methods are designed for, and compared the results to that of our method in its natural setting, i.e. patterns belonging to a subspace. As seen from Fig. 7 the proposed method in this paper could achieve very small recall errors while having a much larger pattern retrieval capacity at the same time.

We then applied the methods of [3] and [19] to the dataset in which patterns belong to a subspace. Interestingly, the performance of the method in [3] deteriorates while that of [19] becomes better in certain cases (e.g. for $C = 2000$). However, neither of the approaches can achieve the small error rates obtained by the method proposed in this paper. Thus, even though redundancy and structure exists in this particular set of patterns, the mentioned approaches could not exploit it to achieve better pattern retrieval capacities or smaller recall error rates.

VIII. Conclusions and Future Work

In this paper, we proposed a neural associative memory which is capable of exploiting inherent redundancy in input patterns to enjoy an exponentially large pattern retrieval capacity. Furthermore, the proposed method uses simple iterative algorithms for both learning and recall phases which makes gradual learning possible and maintain rather good recall performances. The convergence of the proposed learning algorithm was proved using techniques from stochastic approximation. We also analytically investigated the performance of the recall algorithm by deriving an upper bound on the probability of recall error as a function of input noise. Our simulation results confirms the consistency of the theoretical results with those obtained in practice, for different network sizes and learning/recall parameters.

Improving the error correction capabilities of the proposed network is definitely a subject of our future research. We have already started investigating this issue and proposed a different network structure which reduces the error correction probability by a factor of 10 in many cases [28]. In [29] we proposed a more robust recall algorithms which achieves linear error correction capabilities.

Extending this method to capture other sorts of redundancy, i.e. other than belonging to a subspace, will be another topic which we would like to explore in future.

Finally, considering some practical modifications to the learning and recall algorithms is of great interest. One good example is simultaneous learn and recall capability, i.e. to have a network which learns a subset of the patterns in the subspace and move immediately to the recall phase. Now during the recall phase, if the network is given a noisy version of the patterns previously memorized, it eliminates the noise. However, if the given pattern is new and had not learned before, the network adjusts the weights in order to learn this pattern as well. Such model is of practical interest and closer to real-world neuronal networks.

Acknowledgment

The authors would like to thank Prof. Wulfram Gerstner and his lab members, as well as Dr. Amin Karbasi for their helpful comments and discussions. This work was supported by Grant 228021-ECCSciEng of the European Research Council.

Appendix A
Average Neighborhood Size

In this appendix, we find an expression for the average neighborhood size for erroneous nodes, $S_t = \mathbb{E}(|\mathcal{N}(\mathcal{E}_t)|)$. To this end, we assume the following procedure for constructing a right-irregular bipartite graph:

- In each iteration, we pick a variable node x with a degree randomly determined according to the given degree distribution.
- Based on the given degree d_x, we pick d_x constraint nodes uniformly at random *with replacement* and connect x to the constraint node.
- We repeat this process n times, until all variable nodes are connected.

Note that the assumption that we do the process with replacement is made to simplify the analysis.

Now we are interested in finding the average number of constraint nodes in each round of the above procedure. With some abuse of notations, let S_e denote the number of constraint nodes connected to pattern nodes in round e. We write S_e recursively in terms of e as follows:

$$\begin{aligned} S_{e+1} &= \mathbb{E}_{d_x}\Big(\sum_{j=0}^{d_x}\binom{d_x}{j}\left(\frac{S_e}{m}\right)^{d_x-j}\left(1-\frac{S_e}{m}\right)^{j}(S_e+j)\Big) \\ &= \mathbb{E}_{d_x}(S_e + d_x(1-S_e/m)) \\ &= S_e + \bar{d}(1-S_e/m), \end{aligned} \tag{16}$$

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

$$S_t = m\left(1-(1-\frac{\bar{d}}{m})^{|\mathcal{E}_t|}\right) \tag{17}$$

In order to verify the correctness of the above analysis, we have performed some simulations for different network sizes and degree distributions obtained from the graphs returned by the learning algorithm. We generated 2000 random graphs and calculated the average neighborhood size in each iteration. The result for $n = 200$, $m = 100$ is shown in Figure 8, where the average neighborhood size in each iteration is illustrated and compared with theoretical estimations given by equation (17). From the figure, it is obvious that the theoretical value is a very good approximation of the simulation results.

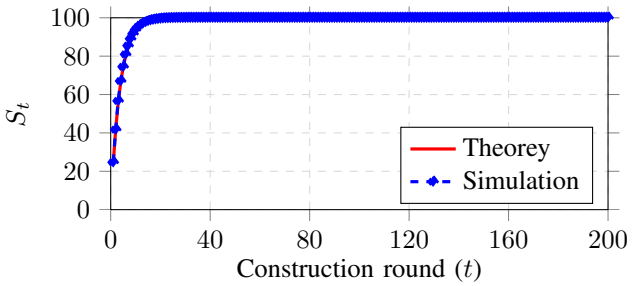

Fig. 8. The theoretical estimation and simulation results for the average neighborhood size of irregular graphs with a given degree-distribution for $n = 200$, $m = 100$ and over 2000 random graphs.

REFERENCES

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 79, no. 8, pp. 2554–2558, 1982.

[2] S. S. Venkatesh and D. Psaltis, "Linear and logarithmic capacities in associative neural networks," *IEEE Trans. Inf. Theor.*, vol. 35, no. 3, pp. 558–568, Sep. 1989.

[3] S. Jankowski, A. Lozowski, and J. M. Zurada, "Complex-valued multistate neural associative memory." *IEEE Trans. Neural Netw. Learning Syst.*, vol. 7, no. 6, pp. 1491–1496, 1996.

[4] M. K. Muezzinoglu, C. Guzelis, and J. M. Zurada, "A new design method for the complex-valued multistate hopfield associative memory," *IEEE Trans. Neur. Netw.*, vol. 14, no. 4, pp. 891–899, Jul. 2003.

[5] A. H. Salavati, K. R. Kumar, M. A. Shokrollahi, and W. Gerstnery, "Neural pre-coding increases the pattern retrieval capacity of hopfield and bidirectional associative memories," in *Proc. IEEE Int. Symp. Inf. Theor. (ISIT)*, 2011, pp. 850–854.

[6] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *IEEE Trans. Neur. Netw.*, vol. 22, no. 7, pp. 1087–1096, 2011.

[7] T. Richardson and R. Urbanke, *Modern Coding Theory*. New York, NY, USA: Cambridge University Press, 2008.

[8] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 379, pp. 948–958, 1948.

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

[10] K. R. Kumar, A. H. Salavati, and M. A. Shokrollahi, "Exponential pattern retrieval capacity with non-binary associative memory," in *IEEE Inf. Theor. workshop (ITW)*, Oct 2011, pp. 80–84.

[11] S. Hoory, N. Linial, A. Wigderson, and A. Overview, "Expander graphs and their applications," *Bull. Amer. Math. Soc.*, vol. 43, no. 4, pp. 439–561, 2006.

[12] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[13] J. Chen, H. Guo, W. Wu, and W. Wang, "imecho: an associative memory based desktop search system," in *Proc. ACM Conf. Information and knowledge management*, 2009, pp. 731–740.

[14] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley & Sons, 1949.

[15] D. J. Amit, H. Gutfreund, and H. Sompolinsky, "Storing infinite numbers of patterns in a spin-glass model of neural networks," *Phys. Rev. Lett.*, vol. 55, pp. 1530–1533, Sep 1985.

[16] J. Hertz, R. G. Palmer, and A. S. Krogh, *Introduction to the Theory of Neural Computation*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991.

[17] J. Komlós and R. Paturi, "Effect of connectivity in an associative memory model," *J. Comput. Syst. Sci.*, vol. 47, no. 2, pp. 350–373, 1993.

[18] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 81, no. 10, pp. 3088–3092, 1984.

[19] D.-L. Lee, "Improvements of complex-valued Hopfield associative memory by using generalized projection rules." *IEEE Trans. Neural Netw.*, vol. 17, no. 5, pp. 1341–1347, 2006.

[20] C. Berrou and V. Gripon, "Coded Hopfield networks," in *Int. Symp. Turbo Codes & Iterative Info. Processing (ISTC)*, Sep 2010, pp. 1–5.

[21] R. H. Gold, "Optimal binary sequences for spread spectrum multiplexing," *IEEE Trans. Inf. Theor.*, vol. 13, no. 4, pp. 619–621, Sep. 1967.

[22] S. S. Venkatesh, "Connectivity versus capacity in the hebb rule," in *Theoretical Advances in Neural Computation and Learning*. Springer, 1994, pp. 173–240.

[23] P. Peretto and J. J. Niez, "Long term memory storage capacity of multiconnected neural networks," *Biol. Cybern.*, vol. 54, no. 1, pp. 53–64, 1986.

[24] B. Kosko, "Bidirectional associative memories," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 18, no. 1, pp. 49–60, 1988.

[25] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[26] P. Tavan, H. Grubmüller, and H. Kühnel, "Self-organization of associative memory and pattern classification: recurrent signal processing on topological feature maps," *Bio. cyber.*, vol. 64, no. 2, pp. 95–105, 1990.

[27] S. Chartier, G. Giguere, D. Langlois, and R. Sioufi, "Bidirectional associative memories, self-organizing maps and k-winners-take-all: uniting feature extraction and topological principles," in *Int. Joint Conf. Neur. Net. (IJCNN)*, 2009, pp. 503–510.

[28] A. H. Salavati and A. Karbasi, "Multi-level error-resilient neural networks," in *Proc. Int. Symp. Inf. Theor. (ISIT)*, 2012, pp. 1064–1068.

[29] A. Karbasi, A. H. Salavati, and A. Shokrollahi, "Iterative learning and denoising in convolutional neural associative memories," in *Proc. Int. Conf. on Machine Learning (ICML)*, ser. ICML '13, Jun. 2013, to appear.

[30] L. Xu, A. Krzyzak, and E. Oja, "Neural nets for dual subspace pattern recognition method," *Int. J. Neur. Syst.*, vol. 2, no. 3, pp. 169–184, 1991.

[31] E. Oja and T. Kohonen, "The subspace learning algorithm as a formalism for pattern recognition and neural networks," in *IEEE Int. Conf. Neur. Netw.*, vol. 1, Jul 1988, pp. 277–284.

[32] E. J. Candès and T. Tao, "Near-optimal signal recovery from random projections: Universal encoding strategies?" *IEEE Trans. Inform. Theor.*, vol. 52, no. 12, pp. 5406–5425, 2006.

[33] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 106, no. 45, pp. 18 914–18 919, 2009.

[34] J. A. Tropp and S. J. Wright, "Computational methods for sparse solution of linear inverse problems," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 948–958, 2010.

[35] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *Math. Analysis and Applications*, vol. 106, pp. 69–84, 1985.

[36] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, D. Saad, Ed. Cambridge Univ. Press, 1998.

[37] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inf. Theor.*, vol. 42, pp. 1710–1722, 1996.

[38] S. Jafarpour, W. Xu, B. Hassibi, and A. R. Calderbank, "Efficient and robust compressed sensing using optimized expander graphs," *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4299–4308, 2009.

[39] K. R. Kumar, A. H. Salavati, and M. A. Shokrollahi, "A non-binary associative memory with exponential pattern retrieval capacity and iterative learning," http://arxiv.org/abs/1302.1156, Ecole Polytechnique Federale de Lausanne (EPFL), Tech. Rep., Feb. 2013.