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

K. Raj Kumar, Amir Hesam Salavati,and Amin Shokrollahi
Laboratoire d'algorithmique (ALGO)
Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland
E-mail: {raj.kumar,hesam.salavati,amin.shokrollahi}@epfl.ch

***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 given patterns using a network of neurons whose states assume values from a finite number of non-negative integer levels. Later, the same network should be able to recall a previously memorized pattern from its noisy version. Prior works 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. We show that if the given patterns have a suitable structure, i.e. come from a sub-space of the set of all possible patterns, we can have exponential pattern retrieval capacities in terms of the length of the patterns. Another example is when the patterns have strong minor components. We will use this minor components (or the basis vectors of the patterns null space) to both increase the pattern retrieval capacity and error correction capabilities.

An iterative algorithm is proposed for the learning phase. In addition, two simple neural update algorithms are presented for the recall phase and using analytical results and simulations, we show that the suggested methods can tolerate a fair amount of errors in the input.

I. INTRODUCTION

Neural associative memory is a particular class of neural memory capable of memorizing (learning) a number of given patterns and recalling them later in presence of noise, i.e. retrieve the correct memorized pattern from a given noisy version. Since 1982 and starting by the seminal work of Hopfield [4], various artificial neural networks have been designed to mimic the task of the neuronal associative memory (see for instance [13], [14], [10], [12], [18]).

In essence, the neural associative memory problem is very similar to the one faced in communication systems where the goal is to reliably transmit and efficiently recover a set of patterns (so called codewords) over a noisy channel. In both cases, we have to learn a set of patterns (codebook), and retrieve the correct one form a noisy version. More interestingly, the techniques used to implement an artificial neural associative memory looks very similar to some of the methods used in codes on graphs to design reliable and efficient codes.

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, all modern codes are capable of reliably transmitting 2^{rn} codewords over a noisy channel, with $0 < r < 1$ being the code rate [20]. In many cases, one can also obtain the optimal r, i.e. the largest possible value that permits the almost sure recovery of transmitted codewords from the corrupted received versions.

In current neural associative memories, however, with a network of size n one can only memorize $O(n)$ binary patterns of length n [11], [13]. 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 (with size $O(n)$ off course) (e.g., [4], [13], [14], [10]). Therefore, they provide us 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 memorization patterns or improve error correction during the recall phase. This is in sharp contrast to coding techniques where one designs codewords such that they have 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 gap between the coding techniques and neural associative memories by proposing a neural network which exploits inherent structure of the input patterns in order to increase the pattern retrieval capacity of the associative memory from $O(n)$ to $O(a^n)$ with $a > 1$. More specifically, the proposed neural network is capable of learning and reliably recalling given patterns when they come from a subspace with dimension $k < n$. Note that although the proposed model does not have the generality of traditional associative memories, such as Hopfield network [4], in cases where there is some input redundancy it enables us to boost the capacity by a great extent. While in same situations, traditional associative memories will still have linear pattern retrieval capacity.

In [2], we explained some preliminary results in which two efficient recall algorithms were proposed for the case where the neural graph was expander. 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 in order to be used later in the recall phase.

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 problem. We explain the proposed learning algorithm in section III. Section IV is dedicated to the recall algorithm and analytically investiagting its performance in retrieving corrupted patterns. In section V we address the pattern retrieval capacity and show it can be exponential in n. Simulation results are discussed in section VI. Finally, section VII concludes the paper and discusses future research topics.

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 $\mathcal{S} = \{0, 1, \ldots, S-1\}$. A natural way of interpreting this model is to think of the integer states as the short-term firing rate of neurons. In other words, the state of a neuron in this model indicates the number of spikes fired by the neuron in a fixed short time interval.

Like in other neural networks, neurons can do only 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 denotes the weight of the input link from the i^{th} neighbor.
2) It updates its state as $x = f(h)$, where $f : \mathbb{R} \to \mathcal{S}$ is a possibly non-linear function from the field of real numbers $\mathbb{R}$ to $\mathcal{S}$.

B. The Problem

We assume to be given C vectors of length n with integer-valued entries belonging to $\mathcal{S}$. Furthermore, we assume these patterns belong to a subspace with dimension $k \leq n$. Note that if $k = n$, then we are back to the original associative memory problem. However, if $k < n$, we will end up with much larger pattern retrieval capacities, as will be shown later.

We would like to memorize these patterns by finding a set of non-zero vectors $w_1, \ldots, w_m \in \mathbb{R}^n$ that are orthogonal to the set of given patterns. Furthermore, we are interested in rather sparse vectors. Denoting pattern μ by x^μ, we can formulate the problem of finding *one* such vector w as:

$$\min \sum_{\mu=1}^{C} |x^\mu \cdot w|^2 \tag{1a}$$

subject to

$$\|w\|_0 \leq q \tag{1b}$$

$$\|w\|_2^2 \geq \epsilon \tag{1c}$$

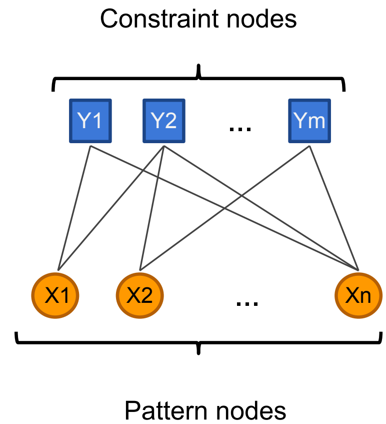

Fig. 1. A bipartite graph that represents the constraints on the training set.

where $\cdot$ represents the inner-product, $q \in \mathbb{N}$ determines the degree of sparsity and $\epsilon \in \mathbb{R}^+$ prevents the all-zero solution.

By repeating the above problem for different vectors w, we will have $m = n - k$ sparse vectors that form the basis for the null space of the patterns which would like to learn[1]. Therefore, the inherent structure of the patterns are captured in the obtained null-basis, denoted by the matrix $W \in \mathbb{R}^{m \times n}$, which can be interpreted as the adjacency matrix of a neural graph. The overall network model is shown in Figure 1.

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 $x_1, x_2, \ldots, x_n$ are initialized with the given input (noisy) pattern. Here, we assume that the noise is integer valued and additive[2]. 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 the pattern x^μ. Any algorithm designed for this purpose should be simple enough to be implemented by neurons. Therefore, we must find a simple algorithm capable of eliminating noise using only neural operation.

Before presenting our solution, we briefly overview the literature and explain the relations between the previous works and the presented approach in this paper.

C. Related Works

Designing a neural associative memory has been an active field for the past three decades. Hopfield was the first to design an artificial neural associative memory in his seminal work back in 1982 [4]. The so-called Hopfield network is inspired by Hebbian learning [8] and is composed of binary-valued (± 1) neurons, which together are able to memorize a certain number of patterns. The pattern retrieval capacity of a Hopfield network of n neurons was derived later by Amit et al. [6]

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

[2] It must be mentioned that neural states below 0 and above $S-1$ will be set to 0 and $S-1$, respectively.

and shown to be $0.13n$, under vanishing bit error probability requirement. Later, McEliece et al. proved that the capacity of Hopfield networks under vanishing pattern error probability requirement is $O(n/\log(n))$ [11].

In addition to neural networks with gradual learning capability, offline methods have also been used to design neural associative memories. For instance, in [13] the authors calculate the weight matrix using the pseudo-inverse rule [7] in an offline manner and for a given set of patterns. 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 to the $n/\log(n)$ scaling of the pattern retrieval capacity in [11], it comes at the price of much higher computational complexity and the lack of gradual learning ability.

While the connectivity graph of a Hopfield network is a complete graph, Kmlos et al. [9] 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. Komlos and Paturi have considered a network in which each neuron is connected to d other neurons, i.e., a d-regular network. Assuming that the network graph satisfies certain connectivity measures, they prove that it is possible to store a linear number of *random* patterns (in terms of d) with vanishing bit error probability or $C = O(d/\log n)$ random patterns with vanishing pattern error probability. Furthermore, they show that in spite of the capacity reduction, the error correction capability remains the same as the network can still tolerate a number of errors which ic linear in n.

It is also known that the capacity of neural associative memories could be enhanced if the patterns are of *low-activity* nature, in the sense that at any time instant many of the neurons are silent [7]. However, even these schemes fail when required to correct a fair amount of erroneous bits as the information retrieval is not better compared to that of normal networks.

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 [5]. In [14] 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. [10] 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 [15].

Given that even very complex offline learning methods can not improve the capacity of binary or multi-sate 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 deigned to be able to memorize any random set of patterns. Pioneering this prospect, Gripon and Berrou [18] 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 [12], 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 [22]. The network itself is very similar to that of Hopfield, with a slightly modified weighting rule. Therefore, similar to a Hopfield network, the complexity of the learning phase is small. However, the authors failed to increase the pattern retrieval capacity beyond n and it was shown that the pattern retrieval capacity of the proposed model is $C = n$, while being able to correct a fair number of erroneous input bits.

In contrast to the pairwise correlation of the Hopfield model [4], Peretto et al. [17] deployed *higher order* neural models: the models in which the state of the neurons not only depends on the state of their neighbors, but also on the correlation among them. Under this model, they showed that the storage capacity of a higher-order Hopfield network can be improved to $C = O(n^{p-2})$, where p is the degree of correlation considered. The main drawback of this model is again 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 [2]. 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 $a > 1$, closing the gap between the pattern retrieval capacities achieved in neural networks and that of coding techniques. The main drawbacks in the proposed approach were the lack of a learning algorithm as well as the expansion assumption on the neural graph.

In this paper, we focus on extending the results described in [2] in several directions: first, we will 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 investigates its performance as a function of input noise and network model. Finally, we discuss some variants of the error correcting method which achieve better performance marks in practice.

It worth mentioning that an extension of this approach to a multi-level neural network is considered in [29]. 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 contrast, the learning algorithm proposed in this paper is capable of learning patterns gradually as they are presented to the network one by one.

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 a set of patterns in the training set (i.e., $Wx^\mu = 0$) using simple neural learning rules (we refer the interested readers to [3] and [16]). 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 [21], in which the measurement matrix is given by the big patterns matrix $\mathcal{X}_{C\times n}$ and the set of measurements are the constraints we look to satisfy, denoted by the tall vector b, which for simplicity reasons we assume to be all zero. Thus, we are interested in finding a sparse vector w such that $\mathcal{X}w = 0$. Nevertheless, many decoders proposed in this area are very complicated and cannot be implemented by a neural network using simple neuron operations. Some exceptions are [1] and [19] which are closely related to the learning algorithm proposed in this paper.

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 [28] to learn the null-space basis of the subspace defined by the patterns. In fact, a very similar algorithm is also used in [3] for the same purpose. However, since we need the basis vectors to be sparse (due to requirements of the algorithm used in the recall phase), we add an additional term to penalize non-sparse solutions during the learning phase.

Another difference with the proposed method and that of [3] is that the learning algorithm proposed in [3] yields dual vectors that form an orthogonal set. Although one can easily extend our suggested method to such a case as well, we find this requirement unnecessary in our case. This gives us the additional advantage to make the algorithm *parallel* and *adaptive*. Parallel in the sense that we can design an algorithm to learn one constraint and repeat it several times in order to find all of the 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), one can easily identify 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.

A. Overview of the proposed algorithm

In order to develop a simple iterative algorithm, we formulate the problem in an optimization framework. The problem to find a constraint vector w is given by equation (1). However, instead of tackling problem (1) directly, we make a slight modification and consider the following optimization problem:

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

subject to:

$$\|w\|_2 = 1 \tag{2b}$$

In the above problem, we have replaced the constraint $\|w\|_0 \leq q$ with a penalty function $g(w)$ since $\|.\|_0$ is not easy to handle analytically. The function $g(w)$ is chosen such that it favors sparsity. 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 [1], [19]. In this paper, we consider the function

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

where σ is chosen appropriately. Intuitively, $\tanh(\sigma w_i^2)$ approximates $|\text{sign}(w_i)|$ in ℓ_0-norm. Therefore, the larger σ is, the closer $g(w)$ will be to $\|.\|_0$. By calculating the derivative of the objective function, and by considering the update due to each randomly picked pattern x^μ, we will get the following iterative algorithm:

$$y(t) = x(t) \cdot w(t) \tag{3a}$$

$$\tilde{w}(t+1) = w(t) - \alpha_t \left(2y(t)x(t) + \lambda\Gamma(w(t))\right) \tag{3b}$$

$$w(t+1) = \frac{\tilde{w}(t+1)}{\|\tilde{w}(t+1)\|} \tag{3c}$$

In the above equations, t is the iteration index, $x(t)$ 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, $\Gamma(w) : \mathcal{R}^n \rightarrow \mathcal{R}^n = \nabla g(w)$ is the gradient of the penalty term for non-sparse solutions.

Remark 1. *The i^{th} entry of the function $\Gamma(w(t))) = \nabla g(w(t))$ is*

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

This function has the interesting property that for very small values of $w_i(t)$, $\Gamma_i(w(t)) \simeq 2\sigma_t w_i(t)$ and for relatively larger values of $w_i(t)$, we get $\Gamma_i(w(t)) \simeq 0$ (see Figure 2). Therefore, by proper choice of λ and σ_t, equation (3b) suppresses small entries of $w(t)$ by pushing them towards zero. In other words, this function favors sparser results.

Following the same approach as [28] and assuming α_t to be small enough such that equation (3c) can be expanded as powers of α_t, we can approximate equation (3) with the following simpler version:

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

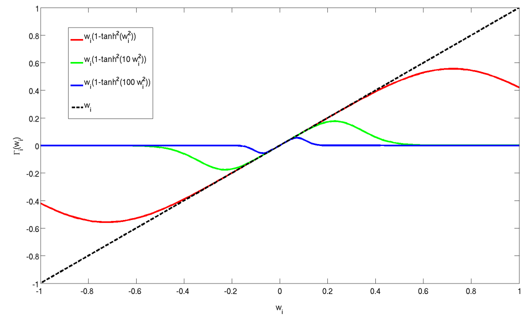

Fig. 2. The sparsity penalty $\Gamma_i(w_i)$, which suppresses small values of the i^{th} entry of w in each iteration as a function of w_i and σ. Note that the normalization constant 2σ has been omitted here to make comparison with function $f = w_i$ possible.

Algorithm 1 Iterative Learning

Input: Set of patterns $x^\mu \in \mathcal{X}$ with $\mu = 1, \dots, C$, 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 $w(t+1) = w(t) - \alpha_t y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|^2} \right) - \alpha_t \lambda \Gamma(w(t))$.
    $t \leftarrow t + 1$.
  end while

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

In the above approximation, we also omitted the term $\alpha_t \lambda \left(w(t) \cdot \Gamma(w(t)) \right) w(t)$ since $w(t) \cdot \Gamma(w(t))$ would be negligible for large σ_t's. In fact it tends to zero as σ_t grows with t.

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 the data vector $x(t)$ has some projection over $w(t)$ then the weight vector is updated towards the direction to reduce this projection.

Since we are interested in finding m basis vectors, we have to do the above procedure m times in parallel.

Remark 2. *Although we are interested in finding a sparse graph, note that too much sparseness is not desired. Because we are going to use the feedback sent by the constraint nodes to eliminate input noise at pattern nodes. Now if the graph is too sparse, the number of feedbacks received by each pattern node is too small to be relied upon. Therefore, we must adjust the penalty coefficient λ such that resulting neural graph is* rather *sparse. In the section on experimental results, we compare the error correction performance for different choices of λ.*

B. Convergence analysis

In order to prove that Algorithm 1 converges to the proper solution, we use results from statistical learning. More specifically, we benefit from the convergence of Stochastic Gradient Descent (SGD) algorithms [25]. To prove the convergence, let $E(w) = \sum_\mu |x^\mu \cdot w|^2$ be the cost function we would like to minimize. Furthermore, let $A = \mathbb{E}\{xx^T | x \in \mathcal{X}\}$ be the corelation patterns for the patterns in the training set. Therefore, due to uniformity assumption for the patterns in the training set, one can rewrite $E(w) = w^T A w$, where we have omitted the normalizing $1/C$ constant for simplicity. finally, denote $A_\mu = x^\mu (x^\mu)^T$. Now consider the following assumptions:

A1. $\|A\|_2 \leq \Upsilon < \infty$ and $\sup_\mu \|A_\mu\|_2 = \|x^\mu\|^2 \leq \zeta < \infty$.
A2. $\alpha_t \geq 0$, $\sum \alpha_t = \infty$ and $\sum \alpha_t^2 < \infty$.

The following lemma porves the convergence of Algorithm 1 to a local minimum w^*.

Lemma 1. *Let assumptions A1 and A2 hold. Then, Algorithm 1 converges to a local minimum w^* for which $\nabla E(w^*) = 0$.*

Proof: To prove the lemma, we use the convergence results in [25] and show that the required assumptions to ensure convergence holds for the proposed algorithm. For simplicity, these assumptions are listed here:

1) The cost function $E(w)$ is three-times differentiable with continuous derivatives. It is also bounded from below.
2) The usual conditions on the learning rates are fulfilled, i.e. $\sum \alpha_t = \infty$ and $\sum \alpha_t^2 < \infty$.
3) The second moment of the update term should not grow more than linearly with size of the weight vector. In other words,
$$E(w) \leq A + B\|w\|_2^2$$
for some constants A and B.
4) When the norm of the weight vector w is larger than a certain horizon D, the opposite of the gradient $-\nabla E(W)$ points towards the origin. Or in other words:
$$\inf \|w\|_2 > D w \cdot \nabla E(w) > 0$$
5) When the norm of the weight vector is smaller than a second horizon F, with $F > D$, then the norm of the update term $(2y(t)x(t) + \lambda \Gamma(w(t)))$ is bounded regardless of $x(t)$. This is usually a mild requirement:
$$\forall x(t) \in \mathcal{X}, \quad \sup_{\|w\|_2 \leq F} \| \left(2y(t)x(t) + \lambda \Gamma(w(t)) \right) \|_2 \leq K_0$$

To start, assumption 1 holds trivially as the cost function is three-times differentiable, with continuous derivatives. Furthermore, $E(w) \geq 0$. Assumption 2 holds because of our choice of the step size α_t, as mentioned in the lemma description.

Assumption 3 ensures that the vector w could not escape by becoming larger and larger. Due to the constraint $\|w\| = 1$, this assumption holds as well.

Assumption 4 holds as well because:

$$\begin{aligned}\mathbb{E}_\mu\left(2A_\mu w + \lambda\Gamma(w)\right)^2 &= 4w^T\mathbb{E}_\mu(A_\mu^2)w + \lambda^2\|\Gamma(w)\|^2 \\ &+ 4\lambda w^T\mathbb{E}_\mu(A_\mu)\Gamma(w) \\ &\leq 4\|w\|^2\zeta^2 + \lambda^2\|w\|^2 + 4\lambda\Upsilon\|w\|^2 \\ &= \|w\|^2(4\zeta^2 + 4\lambda\Upsilon + \lambda^2) \end{aligned} \quad (5)$$

Finally, assumption 5 holds because:

$$\begin{aligned}\|2A_\mu w + \lambda\Gamma(w)\|^2 &= 4w^T A_\mu^2 w + \lambda^2\|\Gamma(w)\|^2 \\ &+ 4\lambda w^T A_\mu \Gamma(w) \\ &\leq \|w\|^2(4\zeta^2 + 4\lambda\zeta + \lambda^2) \end{aligned} \quad (6)$$

Therefore, $\exists F > D$ such that as long as $|w\|^2 < F$:

$$\sup_{\|w\|^2<E} \|2A_\mu w + \lambda\Gamma(w)\|^2 \leq (2\zeta+\lambda)^2 F = \text{constant} \quad (7)$$

Since all necessary assumptions hold for the learning algorithm 1, it converges to a local minimum where $\nabla E(w^*) = 0$. ∎

Next, we prove the desired result, i.e. the fact that in the local minimum, the resulting weight vector is orthogonal to the patterns, i.e. $Aw = 0$.

Theorem 2. *In the local minimum where $\nabla E(w^*) = 0$, the optimal vector w^* is orthogonal to the patterns in the training set, i.e. $Aw^* = 0$.*

Proof: Since $\nabla E(w^*) = 2Aw^* + \lambda\Gamma(w^*) = 0$, we have:

$$w^* \cdot \nabla E(w^*) = 2(w^*)^T Aw^* + \lambda w^* \cdot \Gamma(w^*) \quad (8)$$

The first term is always greater than or equal to zero. Now as for the second term, we have that $|\Gamma(w_i)| \leq |w_i|$ and $\text{sign}(w_i) = \text{sign}(\Gamma(w_i))$, where w_i is the i^{th} entry of w. Therefore, $0 \leq w^* \cdot \Gamma(w^*) \leq \|w^*\|^2$. Therefore, both terms on the right hand side of (8) are greater than or equal to zero. And since the left hand side is known to be equal to zero, we conclude that $(w^*)^T Aw^* = 0$ and $\Gamma(w^*) = 0$. The former means $(w^*)^T Aw^* = \sum_\mu (w^* \cdot x^\mu)^2 = 0$. Therefore, we must have $w^* \cdot x^\mu = 0$, for all $\mu = 1, \ldots, C$. Which simply means the vector w^* is orthogonal to all the patterns in the training set. ∎

Remark 3. *Note that the above theorem only proves that the obtained vector is orthogonal to the data set and says nothing about its degree of sparsity. The reason is that there is no guarantee that the dual basis of a subspace be sparse. The introduction of the penalty function $g(w)$ in problem (4a) only encourages sparsity by suppressing the small entries of w, i.e. shifting them towards zero if they are really small or leaving them intact if they are rather large. And from the fact that $\Gamma(w^*) = 0$, we know this is true as the entries in w^* are either large or zero, i.e. there are no small entries. Our experimental results in section VI show that in fact this strategy works perfectly and the learning algorithm results in sparse solutions.*

C. Avoiding the all-zero solution

Although in problem (2) we have the constraint $\|w\|_2 = 1$ to make sure that the algorithm does not converge to the trivial solution $w = 0$, due to approximations we made when developing the optimization algorithm, we should make sure to choose the parameters such that the all-zero solution is still avoided.

To this end, denote $w'(t) = w(t) - \alpha_t y(t)(x(t) - \frac{y(t)w(t)}{\|w(t)\|^2}$ and consider the following inequalities:

$$\begin{aligned}\|w(t+1)\|^2 &= \|w(t) - \alpha_t y(t)(x(t) - \frac{y(t)w(t)}{\|w(t)\|^2} - \alpha_t\lambda\Gamma(w(t))\|^2 \\ &= \|w'(t)\|^2 + \alpha_t^2\lambda^2\|\Gamma(w(t))\|^2 - 2\alpha_t\lambda\Gamma^T(w(t))w'(t) \\ &\geq \|w'(t)\|^2 - 2\alpha_t\lambda\Gamma^T(w(t))w'(t) \end{aligned} \quad (9)$$

Now in order to have $\|w(t+1)\|^2 > 0$, we must have that $2\alpha_t\lambda|\Gamma(w(t))^T w'(t)| \leq \|w'(t)\|^2$. Given that, $|\Gamma^T(w(t))w'(t)| \leq \|w'(t)\|\|\Gamma(w(t))\|$, it is therefore sufficient to have $2\alpha_t\lambda\|\Gamma(w(t))\| \leq \|w'(t)\|$. On the other hand, we have:

$$\begin{aligned}\|w'(t)\|^2 &= \|w(t)\|^2 + \alpha_t^2 y(t)^2\|x(t) - \frac{y(t)w(t)}{\|w(t)\|^2}\|^2 \\ &\geq \|w(t)\|^2 \end{aligned} \quad (10)$$

As a result, in order to have $\|w(t+1)\|^2 > 0$, it is sufficient to have $2\alpha_t\lambda\|\Gamma(w(t))\| \leq \|w(t)\|$. Finally, since we have $\Gamma(w(t)) \leq w(t)$ (entry-wise), we know that $\|\Gamma(w(t))\| \leq \|w(t)\|$. Therefore, having $2\alpha_t\lambda < 1 \leq \|w(t)\|/\|\Gamma(w(t))\|$ ensures $\|w(t)\| > 0$.

Remark 4. *Interestingly, the above choice for the function $w - \Gamma(w)$ looks very similar to the soft thresholding function (11) introduced in [1] to perform iterative compressed sensing. The authors show that their choice of the sparsity function is very competitive in the sense that one can not get much better results by choosing other thresholding functions. However, one main difference between their work and that of ours is that we enforce the sparsity as a penalty in equation (3b) while they apply the soft thresholding function in equation (11) to the whole w, i.e. if the updated value of w is larger than a threshold, it is left intact while it will be put to zero otherwise.*

$$\eta_t(x) = \begin{cases} x - \theta_t & \textit{if } x > \theta_t; \\ x + \theta_t & \textit{if } x < -\theta_t \\ 0 & \textit{otherwise.} \end{cases} \quad (11)$$

where θ_t is the threshold at iteration t and tends to zero as t grows.

D. Making the Algorithm Parallel

In order to find m constraints, we need to repeat Algorithm 1 several times. Fortunately, we can repeat this process in parallel, which speeds up the algorithm and is more meaningful from a biological point of view as each constraint neuron can act independently of other neighbors. Although doing the algorithm in parallel may result in redundant 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. And the chance of getting redundant constraints reduces if we start from a sparse random initial point. Besides, as long as we have enough distinct constraints, the recall algorithm in the next section works just fine and there is no need to learn all the distinct basis vectors of the null space defined by the training patterns. Therefore, we will use the parallel version to have a faster algorithm in the end.

IV. Recall Phase

In the recall phase, we are going to exploit the facts that the connectivity matrix of the neural graph is sparse and orthogonal to the memorized patterns. Therefore, given a noisy version of the learned patterns, we can use the feedback at the constraint neurons in Fig. 1 to eliminate noise. More specifically, the linear input sum to the constraint neurons is given by 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). We will use sparsity of the neural graph to eliminate z iteratively, as explained in the following.

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 feedback updates its state while the other pattern neurons maintain their current states. The winner-take-all circuitry can be easily added to the neural model shown in Figure 1 using any of the classic WTA methods [7].

The second approach, given by Algorithm 3, is much simpler and in every iteration, each pattern neuron decides locally to update its current state or not. More specifically, if the amount of feedback received by a pattern neuron exceeds a threshold, the neuron updates its state and remains intact otherwise.[3] In both algorithms, the quantity g_j can be interpreted as feedback to pattern neuron x_j from the constraint neurons, where the sign of g_j provides an indication of the sign of the noise that affects x_j, and $|g_j|$ indicates the confidence level in the decision regarding the sign of the noise.

It is worthwhile to mention that the bit-flipping decoding algorithm is very similar to the bit-flipping algorithm of Sipser and Spielman [24] and a similar approach in [23] for compressive sensing methods.

[2]Note that in practice, we replace the condition $h_i = 0$ and $h_i > 0$ with $|h_i| < \varepsilon$ and $h_i > \varepsilon$ for some small positive number ε.

[3]Note that in order to maintain the current value of a neuron in case no input feedback is received, we can add self-loops to pattern neurons in Figure 1. These self-loops are not shown in the figure to make it more clear.

Algorithm 2 Recall Algorithm: Winner-Take-All

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

for $t = 1 \to t_{\max}$ **do**
  Forward iteration: Calculate the weighted input sum $h_i = \sum_{j=1}^n W_{ij}^b x_j$, for each constraint neuron y_i and set:
$$y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise[2]} \end{cases}.$$
  Backward iteration: Each neuron x_j with degree d_j computes
$$g_j = \frac{\sum_{i=1}^m W_{ij}^b y_i}{d_j}.$$
  Find
$$j^* = \arg\max_j |g_j|.$$
  Update the state of winner j^*: If $g_{j^*} \neq 0$, then set $x_{j^*} = x_{j^*} + \text{sign}(g_{j^*})$.
  $t \leftarrow t + 1$
end for

Algorithm 3 Recall Algorithm: Bit-Flipping

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

for $t = 1 \to t_{\max}$ **do**
  Forward iteration: Calculate the weighted input sum $h_i = \sum_{j=1}^n W_{ij}^b x_j$, for each neuron y_i and set:
$$y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise[2]} \end{cases}.$$
  Backward iteration: Each neuron x_j with degree d_j computes
$$g_j = \frac{\sum_{i=1}^m W_{ij}^b y_i}{d_j}.$$
  Update the state of each pattern neuron j according to $x_j = x_j + \text{sign}(g_j)$ only if $|g_j| > \varphi$.
  $t \leftarrow t + 1$
end for

Remark 5. *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: they are based on counting the fraction of received input feedbacks 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. As a result, in the backward iteration all the pattern neuron receive feedback from their* *neighbors and it is impossible to tell which of the pattern neuron is the noisy one.*

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

B. Performance analysis

In order to find out analytical estimations on the recall probability of error, we assume that the connectivity graph W is sparse and the degree distribution for pattern nodes is given, which is determined after the learning phase is finished. Furthermore, we consider an ensemble of random neural graphs with a given degree distribution for pattern nodes and investigate the average error performance. 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 fo having a number of real-valued vectors add up to zero is negligible.

We do the analysis only for the bit-flipping algorithms since if we choose the bit-flipping update threshold $\varphi = 1$, roughly speaking, we will have the winner-take-all algorithm.[4]

When W is also an expander graph, we might get better performances in the recall phase and the analysis for this case can be found in appendix C and in [2]. However, since it is very difficult, if not impossible in certain cases, to make a graph expander during an iterative learning method, we do the analysis for sparse neural graphs only.

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 other 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.

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

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

[4] 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.

We start the analysis finding a explicit relationships and upper bounds on the average of P_{e_1} and P_{e_2} 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 required expressions on the average bit error probability as a function of the number of noisy input nodes, $|\mathcal{E}_0|$. Having found the average bit error probability, we can easily bound the block error probability for the recall algorithm.

1) Error probability - type 1: To begin, let P_1^x be the probability that a node $x \in \mathcal{C}_t$ with degree d_x updates its state. We have:

$$P_1^x = \Pr\{\frac{|\mathcal{N}(\mathcal{E}_t) \cap \mathcal{N}(x)|}{d_x} \geq \varphi\} \tag{12}$$

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 by

$$P_1^x = \sum_{i=\lceil \varphi d_x \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t}{m}\right)^i \left(1 - \frac{S_t}{m}\right)^{d_x - i} \tag{13}$$

As a result of the above equations, we have:

$$P_{e_1} = \mathbb{E}_{d_x}(P_1^x) \tag{14}$$

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

$$P_2^x = \sum_{i=\lceil d_x/2 \rceil}^{d_x} \binom{d_x}{i} \left(\frac{S_t^*}{m}\right)^i \left(1 - \frac{S_t^*}{m}\right)^{d_x - i} \tag{15}$$

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

Therefore, we will have:

$$P_{e_2} \leq \mathbb{E}_{d_x}(P_2^x) \tag{16}$$

Combining equations (14) and (16), the symbol error probability at iteration t would be

$$\begin{aligned} P_b(t) &= \Pr\{x \in \mathcal{C}_t\} P_{e_1} + \Pr\{x \in \mathcal{E}_t\} P_{e_2} \\ &= \frac{n - |\mathcal{E}_t|}{n} P_{e_1} + \frac{|\mathcal{E}_t|}{n} P_{e_2} \end{aligned} \tag{17}$$

where $\bar{P}_i^x = \mathbb{E}_{d_x}\{P_i^x\}$.

And finally, the average block error rate is given by the probability of at least one pattern node x makes a mistake. Therefore:

$$P_e(t) = 1 - (1 - P_b(t))^n \tag{18}$$

Equation (18) gives the probability of making a mistake in iteration t. Therefore, we can bound the overall probability of error as

$$\begin{aligned} P_E &\leq \prod_t P_e(t) \\ &\leq P_e(1) \end{aligned} \tag{19}$$

In other words, if we made a mistake in the first round, we assume everything will go wrong till the end. Obviously, this bound is not tight as in practice and one might be able to correct errors in later iterations. In fact simulation results confirm this conjecture. However, this bound 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.

Remark 6. *One might hope to further simplify the above inequalities by finding closed form apporximation of equations (13) and (15). This is done in Appendix* **??**. *However, as one expects, this approach leads to very loose and trivial bounds in many cases. Therefore, in our experiments shown in section* **??** *we compare simulation results to the theoretical bound derived using equations (13) and (15).*

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:*

$$S_t = m\left(1 - (1 - \frac{\bar{d}}{m})^{|\mathcal{E}_t|}\right) \tag{20}$$

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

Proof: The proof is given in appendix A. ■

Remark 7. *Although the randomness assumption may not be correct due to the fact that the resulting graphs may not correspond to the dual subspace of the given training set, over all ensembles of random training sets this assumption might hold. Our simulation results in the following confirm this assumption as well. Nevertheless, for a particular realization of the neural graph, one might do better or worse than average.*

C. Some Practical Modifications

Although algorithm 3 is fairly simple and practical, one can make modify it to make it achieve better performances in practice. The trick is to replace the degree of each node x_j with the ℓ_1-norm of the outgoing weights. In otherwords, instead of using $d_j = Vertw_j\|_0$, we use $\|w_j\|_1$, which might be simpler in some cases, as shown by our experimental results in section VI.

One possible reason behind this improvement is that using the ℓ_1-norm instead of the ℓ_0-norm in 3 will result in better differentiation between two vectors that have the same number of non-zero elements, i.e. have equal ℓ_0-norms, but differ from each other in the magnitude of the element, i.e. their ℓ_1-norms differ. Therefore, the network may use this additional information in order to pin out the noisy nodes in each update of the recall algorithm.

V. Pattern Retrieval Capacity

It is interesting to see that, except its obvious influence on the learning time, the number of patterns C does not have any effect in the learning or recall algorithm. Because 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 $0 < r < 1$. By valid we mean 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 $S-1$. 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 random vector $u^\mu \in \mathbb{R}^k$ 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 $S-1$. Obviously, since both u^μ and G have only non-negative entries, all entries in x^μ are non-negative. Therefore, it is the $S-1$ 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

$$S - 1 \geq d^*(\gamma - 1)(\upsilon - 1) \tag{21}$$

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

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 is 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 have $S \geq 11$, i.e. the maximum firing rate of neurons should be 11 to have a pattern retrieval capacity of $C = 2^{rn}$.

Remark 8. *Note that the inequality (21) was obtained for the worst-case scenario and in fact is very loose. Therefore, even if it does not hold, we will still be able to memorize a very large number of patterns since a big portion of the generated vectors x^μ will have entries less than S. These vectors correspond* *to the message vectors u^μ that are "sparse" as well, i.e. do not have all entries greater than zero. The number of such vectors is a polynomial in n, the degree of which depends on the number of non-zero entries in u^μ.*

VI. Simulation Results

A. Simulation Scenario

We have simulated the proposed learning and recall algorithms for three different network sizes $n = 100, 200, 400$, with $k = n/2$ for all cases. For each case, we considered three different set-ups with different values for λ and σ in the learning algorithm 1, and different φ for the bit-flipping recall algorithm 3.

In all cases, we generated 50 training set at random 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 $S = 11$ 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 forever. 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 this subset as the training set.

For each set-up, we performed the learning algorithm and then investigated the average sparsity of the learned constraints over the ensemble of 50 instance. 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 IV-B as well.

B. Learning Phase Results

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

Figure 3 illustrates the percentage of variable nodes with the specified sparsity measure defined as $\rho = \kappa/n$, where κ is the number of non-zero elements. From the figure one notices that as n increases, the weight vectors become sparser.

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 algorithm to correct the errors. A pattern error is declared if the output does not match the correct pattern.

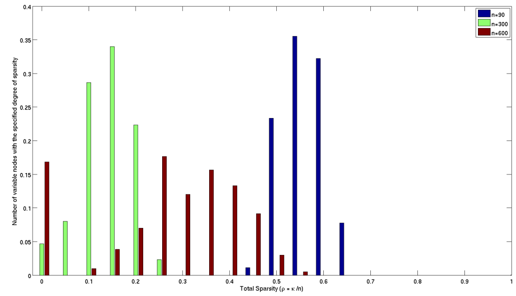

Fig. 3. The percentage of variable nodes with the specified sparsity measure and different values of network sizes. The sparsity measure is defined as $\rho = \kappa/n$, where κ is the number of non-zero elements.

Figure 4 illustrates the pattern error rates for different network sizes. Note that the results are in close match with the theoretical upper bound derived in section IV-B. N

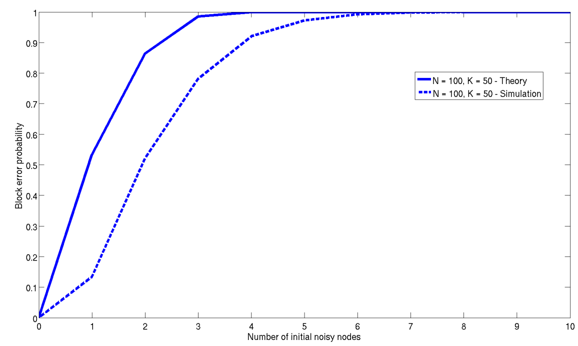

Fig. 4. Pattern error rate against the initial number of erroneous nodes and comparison with theoretical upper bounds

VII. Conclusions and Future Works

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 algorithm 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 [29]. We are working on different structures to obtain even more robust recall algorithms.

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

ACKNOWLEDGMENT

The authors would like to thank Prof. Wulfram Gerstner and his lab, as well as Mr. 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 node, $S_t = \mathbb{E}(|\mathcal{N}(\mathcal{E}_t)|)$. 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. This assumption becomes more exact as n grows.

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

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

$$\begin{aligned} S_{e+1} &= \mathbb{E}_{d_x}\{\sum_{j=0}^{d_x}\binom{d_x}{j}\left(\frac{S_e}{m}\right)^{d_x-j}\left(1-\frac{S_e}{m}\right)^j(S_e+j)\} \\ &= \mathbb{E}_{d_x}\{S_e+d_x(1-S_e/m)\} \\ &= S_e+\bar{d}(1-S_e/m) \end{aligned} \tag{22}$$

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 directly follows from relationship on binomial sums. Noting that $S_1 = \bar{d}$, one obtains:

$$S_t = m\left(1-(1-\frac{\bar{d}}{m})^{|\mathcal{E}_t|}\right) \tag{23}$$

In order to verify the correctness of the above analysis, we have performed some simulations for different network sizes and degree distributions obtained from the graphs returned by the learning algorithm. We generated 100 random graphs and calculated the average neighborhood size in each iteration over these graphs. Furthermore, two different network sizes were considered $n = 100, 200$ and $m = n/2$ in all cases, where n and m are the number of pattern and constraint nodes, respectively. The result for $n = 100, m = 50$ is shown in Figure 5, where the average neighborhood size in each iteration is illustrated and compared with theoretical estimations given by equation(23). Figure 6 shows similar results for $n = 200, m = 100$. In the figure,the dashed line shows the average neighborhood size over these graphs. The solid line corresponds to theoretical estimations. It is obvious that the theoretical value approximates the simulation results rather exactly.

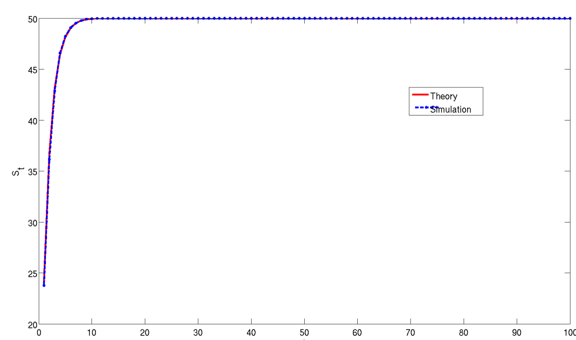

Fig. 5. The theoretical estimation and simulation results for the neighborhood size of irregular graphs with a given degree-distribution for $n = 100, m = 50$ and over 2000 random graphs.

APPENDIX B
EXPANDER GRAPHS

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

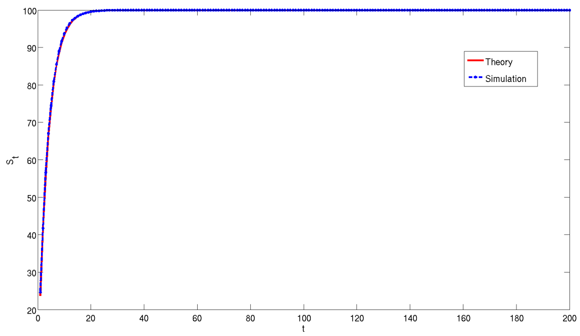

Fig. 6. The theoretical estimation and simulation results for the neighborhood size of irregular graphs with a given degree-distribution for $n = 200$, $m = 100$ and over 2000 random graphs.

Definition 1. *A regular (d_p, d_c, n, m) bipartite graph W is a bipartite graph between n pattern nodes of degree d_p and m constraint nodes of degree d_c.*

Definition 2. *An $(\alpha n, \beta d_p)$-expander is a (d_p, d_c, n, m) bipartite graph such that for any subset $\mathcal{P}$ of pattern nodes with $|\mathcal{P}| < \alpha n$ we have $|\mathcal{N}(\mathcal{P})| > \beta d_p |\mathcal{P}|$ where $\mathcal{N}(\mathcal{P})$ is the set of neighbors of $\mathcal{P}$ among the constraint nodes.*

The following result from [24] shows the existence of families of expander graphs with parameter values that are relevant to us.

Theorem 5. *[24] Let W be a randomly chosen $(d_p, d_c)-$regular bipartite graph between n d_p-regular vertices and $m = (d_p/d_c)$ d_c-regular vertices. Then for all $0 < \alpha < 1$, with high probability, all sets of αn d_p-regular vertices in W have at least*

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

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

The following result from [26] shows the existence of families of expander graphs with parameter values that are relevant to us.

Theorem 6. *Let d_c, d_p, m, n be integers, and let $\beta < 1 - 1/d_p$. There exists a small $\alpha > 0$ such that if W is a (d_p, d_c, n, m) bipartite graph chosen uniformly at random from the ensemble of such bipartite graphs, then W is an $(\alpha n, \beta d_p)$-expander with probability $1 - o(1)$, where $o(1)$ is a term going to zero as n goes to infinity.*

Appendix C
Analysis of the Recall Algorithms for Expander Graphs

A. Analysis of the Winner-Take-All Algorithm

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

Lemma 9. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander and the original number of erroneous neurons are less than $e_{min} = \lfloor \frac{\beta}{1-\beta} \rfloor$, then in each iteration of the winner-take-all algorithm only the corrupted pattern nodes update their value and the other nodes remain intact. For $\beta = 3/4$, the algorithm will always pick the correct node if we have two or fewer erroneous nodes.*

Proof: For simplicity, we restrict our attention to the case $\beta = 3/4$. If we have only one node x_i in error, it is obvious that the corresponding node will always be the winner of the winner-take-all algorithm unless there exists another node that has the same set of neighbors as x_i. However, this is impossible as because of the expansion properties, the neighborhood of these two nodes must have at least $2\beta d_p$ members which for $\beta = 3/4$ is equal to $3d_p/2$. As a result, no two nodes can have the same neighborhood and the winner will always be the correct node.

In the case where there are two erroneous nodes, say x_i and x_j, let Q be the set $\{x_i, x_j\}$ and $\mathcal{N}(Q)$ be the corresponding neighborhood on the constraint nodes side. Furthermore, assume x_i and x_j share $d_{p'}$ of their neighbors so that $|\mathcal{N}(Q)| = 2d_p - d_{p'}$. First of all note that because of the expansion properties and for $\beta = 3/4$:

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

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

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

Now, note that the nodes x_i and x_j will receive some feedback from at least $d_p - d_{p'}$ edges because those are the edges that are uniquely connected to them and noise from the other erroneous nodes cannot cancel them out. Since $d_p - d_{p'} > 3d_p(1-\beta) - d_{p'}$ for $\beta = 3/4$, we conclude that $d_p - d_{p'} > d_{p_\ell}$ which proves that no node outside Q can be picked during the winner-take-all algorithm as long as $|Q| \le 2$ for $\beta = 3/4$. ■

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

Lemma 10. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander and the original number of erroneous neurons is less than $e_{min} = \lfloor \frac{\beta}{1-\beta} \rfloor$, then in each iteration of the winner-take-all algorithm the winner is updated toward reducing the noise.*

Proof: As before, we only focus on the case $\beta = 3/4$. When there is only one erroneous node, it is obvious that all its neighbors agree on the direction of update and the node reduces the amount of noise by one unit.

If there are two nodes x_i and x_j in error, since the number of their shared neighbors is less than $d_p/2$ (as we proved in the last lemma), more than half of their neighbors agree on the direction of update. Therefore, whoever the winner is will be updated to reduce the amount of noise by one unit. ■

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

Theorem 7. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander, then the winner-take-all algorithm is guaranteed to correct at least $e_{min} = \lfloor \frac{\beta}{1-\beta} \rfloor$ positions in error, irrespective of the magnitudes of the errors.*

Proof: The proof is immediate from Lemmas 9 and 10. ■

B. Analysis of the Bit-Flipping Algorithm

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

Theorem 8. *If the constraint matrix W is an $(\alpha n, \beta d_p)$ expander with $\beta > \frac{4}{5}$, then the bit-flipping algorithm with $\gamma = \frac{3}{5}$ is guaranteed to correct at least two positions in error, irrespective of the magnitudes of the errors.*

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

a) Case 1: First consider the case that only one pattern node x_i is in error. Let x_j be any other pattern node, for some $j \neq i$. Let x_i and x_j have $d_{p'}$ neighbors in common. As argued in the proof of Lemma 9, we have that

$$d_{p'} < 2d_p(1-\beta). \tag{24}$$

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

b) Case 2: Now suppose that two distinct nodes x_i and x_j are in error. Let $Q = \{x_i, x_j\}$, and let x_i and x_j share $d_{p'}$ common neighbors. If the noise corrupting these two pattern nodes, denoted by z_i and z_j, are such that $\text{sign}(z_i) = \text{sign}(z_j)$, then both x_i and x_j receive $-\text{sign}(z_i)$ along all d_p edges that they are connected to during the backward iteration. Now suppose that $\text{sign}(z_i) \neq \text{sign}(z_j)$. Then x_i (x_j) receives correct feedback from at least the $d_p - d_{p'}$ edges in $\mathcal{N}(\{x_i\})\backslash Q$ (resp. $\mathcal{N}(\{x_j\})\backslash Q$) during the backward iteration. Therefore, if $d_{p'} < d_p/2$, the direction of update would be also correct and the feedback will reduce noise during the update. However, from equation (24) we know that for $\beta = 4/5$, $d_{p'} \leq 2d_p/5 < d_p/2$. Therefore, the two noisy nodes will be updated towards the correct direction.

Let us now examine what happens to a node x_ℓ that is different from the two erroneous nodes x_i, x_j. Suppose that x_ℓ is connected to d_{p_ℓ} nodes in $\mathcal{N}(Q)$. From the proof of Lemma 9, we know that

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

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

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

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

■

C. Minimum Distance of Patterns

Next, we present a sufficient condition such that the minimum Hamming distance[5] between these exponential number of patterns is not too small. In order to prove such a result, we will exploit the expansion properties of the bipartite graph W; our sufficient condition will be in terms of a lower bound on the parameters of the expander graph.

Theorem 9. *Let W be a $(d_p, d_c, n, m)-$regular bipartite graph, that is an $(\alpha n, \beta d_p)$ expander. Let $\mathcal{X}$ be the set of patterns corresponding to the expander weight matrix W.. If*

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

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

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

$$c_1 W_{i_1} + c_2 W_{i_2} + \cdots + c_d W_{i_d} = 0, \tag{25}$$

where $i_1, \ldots, i_d$ are distinct integers between 1 and n. Let $\mathcal{P}$ denote any set of pattern nodes of the graph represented

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

by W, with $|\mathcal{P}| = d$. As in [23], we divide $\mathcal{N}(\mathcal{P})$ into two disjoint sets: $\mathcal{N}_{unique}(\mathcal{P})$ is the set of nodes in $\mathcal{N}(\mathcal{P})$ that are connected to only one edge emanating from $\mathcal{P}$, and $\mathcal{N}_{>1}(\mathcal{P})$ comprises the remaining nodes of $\mathcal{N}(\mathcal{P})$ that are connected to more than one edge emanating from $\mathcal{P}$. If we show that $\mathcal{N}_{unique}(\mathcal{P}) > 1$ for all $\mathcal{P}$ with $|\mathcal{P}| = d$, then (25) cannot hold, allowing us to conclude that no two patterns with distance d exist. Using the arguments in [23, Lemma 1], we obtain that

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

Hence no two patterns with distance d exist if

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

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

D. Choice of Parameters

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

- From Theorems 6 and 9, the choice of β depends on d_p, according to $\frac{1}{2} + \frac{1}{4d_p} < \beta < 1 - \frac{1}{d_p}$.
- Choose d_c, S, υ, γ so that Theorem 4 yields an exponential number of patterns.
- For a fixed α, n has to be chosen large enough so that an $(\alpha n, \beta d_p)$ expander exists according to Theorem 6, and so that $\alpha n/2 > e_{min} = \lfloor \frac{\beta}{1-\beta} \rfloor$.

Once we choose a judicious set of parameters according to the above requirements, we have a neural associative memory that is guaranteed to recall an exponential number of patterns even if the input is corrupted by errors in two coordinates. Our simulation results will reveal that a greater number of errors can be corrected in practice.

References

[1] D. L. Donoho, A. Maleki, A. Montanari, *Message passing algorithms for compressed sensing*, Proc. Nat. Acad. Sci., Vol. 106, 2009, pp. 1891418919.

[2] K.R. Kumar, A.H. Salavati and A. Shokrollahi, *Exponential pattern retrieval capacity with non-binary associative memory*, Proc. IEEE Information Theory Workshop, 2011.

[3] L. Xu, A. Krzyzak, E. Oja, Neural nets for dual subspace pattern recognition method, Int. J. Neur. Syst., Vol. 2, No. 3, 1991, pp. 169-184.

[4] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proc. Natl. Acad. Sci., Vol. 79, 1982, pp. 2554-2558.

[5] J. J. Hopfield, *Neurons with graded response have collective computational properties like those of two-state neurons*, Proc. Natl. Acad. Sci., Vol. 81, No. 10, 1984, pp. 3088 - 3092.

[6] D. Amit, H. Gutfreund, H. Sompolinsky, *Storing infinite numbers of patterns in a spin-glass model of neural networks*, Physic. Rev. Lett., Vol. 55, 1985, pp. 1530-1533.

[7] J. Hertz, A. Krogh, R. G. Palmer, *Introduction to the theory of neural computation*, USA: Addison-Wesley, 1991.

[8] D. O. Hebb, *The organization of behavior*, New York: Wiley & Sons, 1949.

[9] J. Komlos, R. Paturi, *Effect of connectivity in an associative memory model*, J. Computer and System Sciences, 1993, pp. 350-373.

[10] M. K. Muezzinoglu, C. Guzelis, J. M. Zurada, *A new design method for the complex-valued multistate Hopfield associative memory*, IEEE Trans. Neur. Net., Vol. 14, No. 4, 2003, pp. 891-899.

[11] R. McEliece, E. Posner, E. Rodemich, S. Venkatesh, *The capacity of the Hopfield associative memory*, IEEE Trans. Inf. Theory, Jul. 1987.

[12] A. H. Salavati, K. R. Kumar, W. Gerstner, A. Shokrollahi, *Neural Pre-coding Increases the Pattern Retrieval Capacity of Hopfield and Bidirectional Associative Memories*, To be presented at the IEEE Intl. Symp. Inform. Theory (ISIT - 11), St. Petersburg, Aug 2011.

[13] S. S. Venkatesh, D. Psaltis, *Linear and logarithmic capacities in associative neural networks*, IEEE Trans. Inf. Theory, Vol. 35, No. 3, 1989, pp. 558-568.

[14] S. Jankowski, A. Lozowski, J.M., Zurada, *Complex-valued multistate neural associative memory*, IEEE Tran. Neur. Net., Vol. 1 , No. 6, 1996, pp. 1491-1496.

[15] D. L. Lee, *Improvements of complex-valued Hopfield associative memory by using generalized projection rules*, IEEE Tran. Neur. Net.,Vol. 12, No. 2, 2001, pp. 439-443.

[16] E. Oja, T. Kohonen, *The subspace learning algorithm as a formalism for pattern recognition and neural networks*, Neural Networks, Vol. 1, 1988, pp. 277-284.

[17] P. Peretto, J. J. Niez, *Long term memory storage capacity of multiconnected neural networks*, Biological Cybernetics, Vol. 54, No. 1, 1986, pp. 53-63.

[18] V. Gripon, C. Berrou, *Sparse neural networks with large learning diversity*, IEEE Trans. on Neural Networks, Vol. 22, No. 7, 2011, pp. 10871096.

[19] J. Tropp J, S. J. Wright, *Computational methods for sparse solution of linear inverse problems*, Proc. IEEE, Vol. 98, No. 6, 2010, pp. 948-958.

[20] T. Richardson and R. Urbanke. Modern Coding Theory. Cambridge University Press, 2008.

[21] E. Cands, T. Tao, *Near optimal signal recovery from random projections: Universal encoding strategies?*, IEEE Trans. on Information Theory, Vol. 52, No. 12, 2006, pp. 5406 - 5425.

[22] R. Gold, *Optimal binary sequences for spread spectrum multiplexing*, IEEE Trans. Inf. Theory, Vol. 13, No. 4, 1967, pp. 619-621.

[23] W. Xu, B. Hassibi, *Efficient compressive sensing with deterministic guarantees using expander graphs*, Proc. Inf. Theo. Workshop (ITW), 2007, pp. 414-419.

[24] M. Sipser and D. Spielman, *Expander codes*, IEEE Trans. Inform. Theory, Vol. 42, No. 6, pp. 1710-1722, 1996.

[25] L. Bottou, *Online algorithms and stochastic approximations*, in David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[26] D. Burshtein and G. Miller, *Expander graph arguments for message passing algorithms*, IEEE Trans. Inform. Theory, pp. 782-790, Feb. 2001.

[27] H. J. Kushner, G. G. Yin, *Stochastic approximation algorithms and applications*, Springer Verlag, 1997.

[28] E. Oja, J. Karhunen, *On stochastic approximation of eigenvectors and eigenvalues of the expectation of a random matrix*, J. Math. Analysis and Applications, Vol. 106, No. 1, 1985, pp 6984.

[29] A. H. Salavati, A. Karbasi, *Multi-Level Error-Resilient Neural Networks with Learning* , Submitted to the IEEE Int. Sym. Inf. Theory (ISIT 2012), 2012.