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 networks are famous for their ability to *learn* and *reliably* perform a required task. An important example is the case of (associative) memory where we are asked to memorize (learn) a set of given patterns. Later, corrupted versions of the memorized patterns will be shown to us and we have to return the correct memorized patterns. In essence, this problem is very similar to the one faced in communication systems where the goal is to reliably transmit and efficiently decode a set of patterns (so called codewords) over a noisy channel.

As one would naturally expect, reliability is certainly a very important issue both the neural associative memories and in communication systems. Indeed, the last three decades witnessed many reliable artificial associative neural networks. See for instance [4], [13], [14], [10], [12], [18].

However, despite common techniques and methods deployed in both fields (e.g., graphical models, iterative algorithms, etc), there has been a quantitative difference in terms of another important criterion: the efficiency.ver the past decade, by using probabilistic graphical models in communication systems it has become clear that the number of patterns that can be reliably transmitted and efficiently decoded over a noisy channel is exponential in n, length of the codewords, [20]. However, using current neural networks of size n to memorize a set of *randomly* chosen patterns, the maximum number of patterns that can be reliably memorized scales linearly in n [11], [13].

There are multiple reasons for the inefficiency of the storage capacity of neural networks. First, neurons can only perform simple operations. As a result, most of the techniques used in communication systems (more specifically in coding theory) for achieving exponential storage capacity are prohibitive in neural networks. Second, a large body of past work (e.g., [4], [13], [14], [10]) followed a common assumption that a neural network should be able to memorize *any* subset of patterns drawn randomly from the set of all possible vectors of length n. Although this assumption gives the network a sense of generality, it reduces its storage capacity to a great extent.

An interesting question which arises in this context is whether one can increase the storage capacity of neural networks beyond the current linear scaling and achieve results similar to coding theory. To this end, Kumar et al. [2] suggested a new formulation of the problem where only a suitable set of patterns was considered for storing. This way they could show that the performance of neural networks in terms of storage capacity increases significantly. Following the same philosophy, we will focus on memorizing a random subset of patterns of length n such that the dimension of the training set is $k < n$. In other words, we are interested in memorizing a set of patterns that have a certain degree of *structure* and *redundancy*. We exploit this structure both to increase the number of patterns that can be memorized (from linear to exponential) and to increase the number of errors that can be corrected when the network is faced with corrupted inputs.

The success of [2] is mainly due to forming a bipartite network/graph (as opposed to a complete graph) whose role is to enforce the suitable constraints on the patterns, very similar to the role played by Tanner graphs in coding. More specifically, one layer is used to feed the patterns to the network (so called variable nodes in coding) and the other takes into account the inherent structure of the input patterns

(so called check nodes in coding). A natural way to enforce structures on inputs is to assume that the connectivity matrix of the bipartite graph is orthogonal to all of the input patterns. However, the authors in [2] heavily rely on the fact that the bipartite graph is fully known and given, and satisfies some sparsity and expansion properties. The expansion assumption is made to ensure that the resulting set of patterns are resilient against fair amount of noise. Unfortunately, no algorithm for finding such a bipartite graph was proposed.

Our main contribution in this paper is to relax the above assumptions while achieving better error correction performance. More specifically, we first propose an iterative algorithm that can find a sparse bipartite graph that satisfies the desired set of constraints. We also provide an upper bound on the block error rate of the method that deploys this learning strategy. We then proceed to devise a multi-layer network whose performance in terms of error tolerance improves significantly upon [2] and no longer needs to be an expander.

The remainder of this paper is organized as follows ???

II. Problem Formulation

In contrast to the mainstream work in neural associative memories, we focus on non-binary neurons, i.e., neurons that can assume a finite set of integer values $\mathcal{S} = \{0, 1, \dots, S-1\}$ for their states (where $S > 2$). A natural way to interpret the multi-level states is to think of the short-term (normalized) firing rate of a neuron as its output. Neurons can only perform simple operations. In particular, we restrict the operations at each neuron to a *linear summation* over the inputs, and a possibly non-linear thresholding operation. In particular, a 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 s_i.
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 $\mathcal{S}$.

Neural associative memory aims to memorize C patterns of length n by determining the weighted connectivity matrix of the neural network (*learning phase*) such that the given patterns are stable states of the network. Furthermore, the network should be able to tolerate a fair amount of noise so that it can return the correct memorized pattern in response to a corrupted query (*recall phase*). Among the networks with these two abilities, the one with largest C is the most desirable.

We first focus on learning the connectivity matrix of a neural graph which memorizes a set of patterns having some inherent redundancy. More specifically, we assume to have C vectors of length n with non-negative integer entries, where these patterns form a subspace of dimension $k < n$. We would like to memorize these patterns by finding a set of non-zero vectors $w_1, \dots, w_m \in \mathbb{R}^n$ that are orthogonal to the set of given patterns. Furthermore, we are interested in rather sparse vectors. Putting the training patterns in a matrix $\mathcal{X}_{C\times n}$ and focusing on one such vector w, we can formulate the problem as:

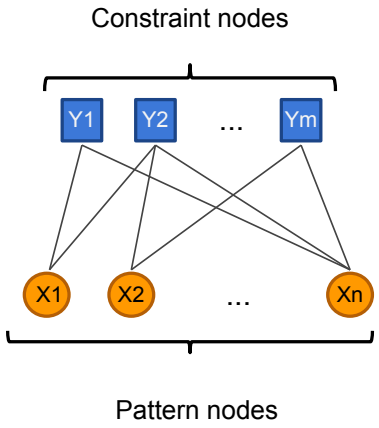

Fig. 1. A bipartite graph that represents the constraints on the training set.

$$\min \|\mathcal{X} \cdot w\|_2 \tag{1a}$$

subject to

$$\|w\|_0 \leq q \quad \text{and} \quad \|w\|_2^2 \geq \epsilon \tag{1b}$$

where $q \in \mathbb{N}$ determines the degree of sparsity and $\epsilon \in \mathbb{R}^+$ prevents the all-zero solution. A solution to the above problem yields a sparse bipartite graph which corresponds to the basis vectors of the null space specified by the patterns in the training set. In other words, the inherent structure of the patterns is captured in terms of m linear constraints on the entries of the patterns x^μ in the training set. It can therefore be described by Figure 1 with a connectivity matrix $W \in \mathbb{R}^{m\times n}$ and a vector $b \in \mathbb{R}^m$ such that $Wx^\mu = b$ for all $\mu = 1, \dots, C$. In this paper, we assume $b = \mathbf{0}$ for simplicity, where $\mathbf{0}$ is the all-zero vector.

In the recall phase, the neural network is fed with noisy inputs. A possibly noisy version of an input pattern is initialized as the states of the pattern neurons $x_1, x_2, \dots, x_n$. Here, we assume that the noise is integer valued and additive[1]. In formula, we have $y = W(x^\mu + z) = Wz$ where z is the noise added to pattern x^μ and we used the fact that $Wx^\mu = \mathbf{0}$. Therefore, one can use $y = Wz$ to eliminate the input noise z. Consequently, we are searching an algorithm that can provably eliminate the effect of noise and return the correct pattern.

Remark 1. *A solution in the learning/recall phase is acceptable only if it can be found by simple operations at neurons.*

Before presenting our solution, we briefly overview the relation between the previous works and the one presented in this paper.

A. Related Works

Designing a neural network capable of learning a set of patterns and recalling them later in presence of noise has been an active topic of research for the past three decades. Inspired by the Hebbian learning rule [8], Hopfield in his seminal

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

work [4] introduced the Hopfield network: an auto-associative neural mechanism of size n with binary state neurons in which patterns are assumed to be binary vectors of length n. The capacity of a Hopfield network under vanishing bit error probability was later shown to be $0.13n$ by Amit et al. [6]. Later on, McEliece et al. proved that the capacity of Hopfield networks under vanishing block error probability requirement is $O(n/\log(n))$ [11].

Similar results were obtained for sparse regular neural network in [9]. Komlos and Paturi extended the results of [11] to neural networks with general connectivity, i.e. those that are not fully interconnected [9]. Their results are especially interesting as physiological data is also in favor of sparsely interconnected neural networks. The authors consider a network in which each neuron is connected to d other neurons, i.e., a d-regular network. Given 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 block 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 linear in n.

It is also known that the capacity of neural associative memories could be enhanced if the patterns are *sparse* 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.

In addition to neural networks capable of learning patterns gradually, in [13], the authors calculate the weight matrix offline (as opposed to gradual learning) using the pseudo-inverse rule [7] which in return help them improve the capacity of a Hopfield network to $n/2$ *random patterns* with the ability of *one bit* error correction. Although this was 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 the ability to learn patterns online (one by one).

Due to the low capacity of Hopfield 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 memories 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 Hopfield networks, a line of recent work has made considerable efforts to exploit the inherent structure of the patterns in order to increase both capacity and error correction capabilities. Such methods either make use of higher order correlations of patterns or focus merely on those patterns that have some sort of redundancy. As a result, they differ from previous methods for which every possible random set of patterns was considered. Pioneering this prospect, Berrou and Gripon [18] achieved considerable improvements in the pattern retrieval capacity of Hopfield networks, by utilizing Walsh-Hadamard sequences. This improvement was made by paying the price of using a decoder based on winner-take-all approach which requires a separate neural network. Therefore, this approach increases 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 with low correlation properties. The network itself is very similar to that of Hopfield, with a slightly modified weighting rule. The proposed methods employ low-complexity learning. Using computer simulations, 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 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 was again the huge computational complexity required in the learning phase.

To address this difficulty while being able to capture higher-order correlations, a bipartite graph inspired from iterative coding theory was introduced in [2]. 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 is the lack of a learning algorithm. The sparsity criterion on the other hand, as it was noted by the authors, is necessary in the recall phase and biologically more meaningful.

In this paper, we address the error correction approach mentioned in [2] and solve the learning problem by proposing an iterative algorithm that identifies a *sparse* weight matrix W. The wight matrix W captures a set of linear constraints such that $Wx^\mu = 0$ for all the patterns x^μ in the training data set, where $\mu = 1, \ldots, C$.

An extension (???) of this approach to a multi-level neural network was considered in [**?**]. There, the novel structure enables better error correction. However, the learning algorithm lacks the online capability.

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] from which we derive our learning algorithm.

III. Learning Phase

In this section, we propose an iterative algorithm capable of learning linear constraints over the set of patterns. These constraints are captured as the set of vectors orthogonal to the given patterns. Therefore, one might naturally think of finding the basis vectors of the null space defined by the set of given patterns. Not surprisingly, there have been neural algorithms which exactly does that. See [3] for instance.

However, since in the recall phase we need a sparse neural graph, we need these null bases be sparse as well, which is not addressed in previous works. Therefore, inspiring from the dual subspace pattern recognition in [3] and a similar approach in [**?**], we propose an iterative learning algorithm which yields sparse vectors orthogonal to the given patterns which the network should memorize.

Another difference with the proposed method and that of [3] is that in [3] the authors design their method such that the resulted dual vectors 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 the constraints. Of course there is always the chance of having two neurons finding the same constraint. Nevertheless, our experimental results show that if we start from distinct random initial points, most of the times the algorithm finds distinct constraints.

And adaptive in the sense that we can determine the number of constraints on-the-go, i.e. start by identifying 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. It also has a nice neural explanation: if the set of given patterns are very simple to memorize, there is no need to identify all the internal constraints as they are so distinct one can easily recall them later even in presence of noise. However, if the set of patterns are Picasso's cubism paintings, obviously we have to spend more time on learning more constraints in order to recall the patterns later from a noisy version.

A. Overview of the proposed algorithm

In order to develop a simple iterative algorithm, we formulate the problem as an optimization framework and then use primal-dual approaches to iteratively find the solution. The problem to find a constraint vector W is given by equation (2).

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

subject to

$$\|w\|_0 \leq q \tag{2b}$$

and

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

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

Therefore, we first relax (2) as follows:

$$\min \sum_{\mu=1}^{C} |x^\mu \cdot w|^2 + \lambda(g(w) - q'). \tag{3a}$$

subject to:

$$\|w\|_2^2 \geq \epsilon \tag{3b}$$

In the above problem, we have approximated the constraint $\|w\|_0 \leq q$ with $g(w) \leq q'$ since $\|.\|_0$ is not a well-behaved function. 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 minimizations. In this paper, we consider the function

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

where σ is chosen appropriately. The larger σ is, the closer $g(w)$ will be to $\|.\|_0$. By calculating the derivative of the objective function and primal-dual optimization techniques we obtain the following iterative algorithm for (3):

$$y(\mu, t) = x^\mu.w(t) \tag{4a}$$

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

$$\lambda_{t+1} = \lambda_t + \gamma(g(w(t)) - q) \tag{4c}$$

In the above equations, t is the iteration index, x^μ and w are vectors of length n, x^μ is the data vector μ, and α_t is a small positive constant. Finally, $f(w) : \mathcal{R}^n \to \mathcal{R}^n = \nabla g(w)$ is the gradient of the penalty term for non-sparse solutions. The term $\frac{y(\mu,t)w(t)}{\|w(t)\|^2}$ is adopted from [3] and takes care of the constraint (2c) as we will show later.

In words, y is the projection of x^μ on the basis vector w. If for a given data vector x^μ, y is equal to zero, namely, the data is orthogonal to the weight vector w, then according to equation (4b) the weight vector will not be updated. However, if the data vector x^μ has some projection over w^μ then the weight vector is updated towards the direction to reduce this

Algorithm 1 Iterative Learning

Input: Set of patterns x^μ with $\mu = 1, \ldots, C$, stopping point p.

Output: w

while $\max_\mu |y(\mu, t)| > p$ **do**

Compute $y(\mu, t) = x^\mu.w(t)$

Update $w(t+1) = w(t) - \alpha_t y(\mu, t) \left(x^\mu - \frac{y(\mu,t)w(t)}{\|w(t)\|^2} \right) - \lambda_t f(w(t))$.

Update $\lambda_{t+1} = \left[\lambda_t + \delta(\epsilon - \|w\|_2^2) \right]$.

$t \leftarrow t + 1$.

end while

projection. The overall learning algorithm for one constraint node y is given by algorithm 1.

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

B. Convergence analysis

In order to prove that the algorithm 1 converges to the proper solution, we rewrite equation (4b) as:

$$w(t+1) = w(t) \left(1 + \alpha (\frac{y(\mu,t)}{\|w(t)\|})^2 \right) - \alpha y(\mu, t) x^\mu - \lambda_t f(w(t)) \tag{5}$$

Now multiplying both sides of the above equation with x^μ we will get:

$$\begin{aligned} y(\mu, t+1) &= y(\mu, t) \left(1 + \alpha \left[(\frac{y(\mu,t)}{\|w(t)\|})^2 - \|x^\mu\|^2 \right] \right) \\ &- \lambda_t x^\mu.f(w(t)) \end{aligned} \tag{6}$$

As a result

$$\begin{aligned} |y(\mu, t+1)| &= |y(\mu, t) \left(1 + \alpha \left[(\frac{y(\mu,t)}{\|w(t)\|})^2 - \|x^\mu\|^2 \right] \right) \\ &- \lambda_t x^\mu.f(w(t))| \\ &\leq |y(\mu, t) \left(1 + \alpha \left[(\frac{y(\mu,t)}{\|w(t)\|})^2 - \|x^\mu\|^2 \right] \right)| \\ &+ \lambda_t |x^\mu.f(w(t))| \end{aligned} \tag{7}$$

Now if we choose the function $f(w(t))$ such that $|f(w(t))| \leq |w(t)|$ and $sign(f(w(t))) = sign(w(t))$ for all values of $w(t)$, then we will have

$$|y(\mu, t+1)| \leq |y(\mu, t)|.|\left(1 + \lambda_t + \alpha \left[(\frac{y(\mu,t)}{\|w(t)\|})^2 - \|x^\mu\|^2 \right] \right)| \tag{8}$$

Therefore, in order to have $|y(\mu, t+1)| \leq y(t)$, we must show pick the coefficients such that

$$|1 + \lambda_t + \alpha \left((\frac{y(\mu,t)}{\|w(t)\|})^2 - \|x^\mu\|^2 \right)| < 1 \tag{9}$$

or equivalently (assuming $\alpha > 0$):

$$\frac{\lambda_t}{\alpha} < \|x^\mu\|^2 - (\frac{y(\mu,t)}{\|w(t)\|})^2 < \frac{2 + \lambda_t}{\alpha} \tag{10}$$

Now noting that $y(\mu, t) = x^\mu.w(t) = \|x^m u\| \|w(t)\| \cos(\theta_{\mu,t})$, we can simplify the above inequality to:

$$\frac{\lambda_t}{\alpha} < \|x^\mu\|^2 \sin^2(\theta_{\mu,t}) < \frac{2 + \lambda_t}{\alpha} \tag{11}$$

This last equation gives us the following inequalities for the value of α:

$$\alpha < \frac{2 + \lambda_t}{\|x^\mu\|^2 \sin^2(\theta_{\mu,t})}, \ \forall \mu \Rightarrow \alpha < \frac{2 + \lambda_t}{\max(\|x^\mu\|^2)} \tag{12a}$$

and

$$\alpha > \frac{\lambda_t}{\|x^\mu\|^2 \sin^2(\theta_{\mu,t})}, \ \forall \mu \tag{12b}$$

The above equations only prove that the algorithm convegres as long as there is a proper relationship between α and λ in every iteration. However, it does not tell us any thing about the degree of sparsity achieved in the end. Nevertheless, simulation results show that we will achieve some good degree of sparsity in the end. See the results section for more details.

C. Some Remarks

There some technical remarks about the approach discussed above:

Remark 2. *In the proposed algorithm above, we chose $g(w)$ such that it approximates the ℓ_0-norm of w. As a result, the i^{th} entry of the function $f(w(t)) = \nabla g(w(t))$ was driven to be $f_i(w(t)) = \partial g(w(t))/\partial w_i(t) = 2\sigma_t w_i(t)(1 - \tanh^2(\sigma_t w_i(t)^2))$. This function has the interesting property that for very small values of $w_i(t)$, $f_i(w(t)) \simeq 2\sigma_t w_i(t)$ and for relatively larger values of $w_i(t)$, we get $f_i(w(t)) \simeq 0$. Therefore, by proper choice of λ_t, equation (4b) suppresses small values of$w(t)$ by pushing them towards zero. In other words, this function favors sparser results.*

*Interestingly, the above choice for the function $f(w)$ looks very similar to the soft thresholding function (**??**) by [1] to perform iterative compressed sensing. The authors show that their choice of the sparsity function is very competetive 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 (4b) while they apply the soft thresholding function in equation (**??**) 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.*

Remark 3. ***Proper choice of** α **and** λ: The above formulas assume that we fix a pattern and iteratively learn it (by increasing the iteration number t). Then, apply another pattern and learn that one (by increasing the pattern index μ). As we have shown above, if α and λ satisfy the set of inequalities (12), then we are pretty sure that $|y(\mu, t+1)| < |y(\mu, t)|$, i.e. the error term for the current pattern μ reduces. However, there is subtlety here as the update along one direction for one pattern may ruin the patterns that we have learned before. So one thing that we can do is to choose α rather small in the hope that update in one direction does not affect the patterns already learnt. However, since according to inequalities (12)*

α and λ are linked together we might want to enforce a bound on λ as well.

One idea to avoid ruining other already learnt patterns is that we make sure that the norm of the update in each iteration is less than a small factor of the norm of the weight vector.Namely,

$$\Delta w_t = \|\alpha_t y(\mu,t)\left(x^\mu - \frac{y(\mu,t)w(t)}{\|w(t)\|^2}\right)\|^2 \leq \beta^2\|w(t)\|^2 \tag{13}$$

where β is a very small constant. In this way, we slow down the convergence but make sure we converge! Expanding the left hand side of inequality (13), we will get

$$\begin{aligned}\Delta w_t &= \|\alpha_t y(\mu,t)\left(x^\mu - \frac{y(\mu,t)w(t)}{\|w(t)\|^2}\right)\|^2 \\ &= \alpha_t^2 y^2(\mu,t)\left(\|x^\mu\|^2 - \frac{y^2(\mu,t)}{\|w(t)\|^2}\right) \\ &= \alpha_t^2 y^2(\mu,t)\|x^\mu\|^2\sin^2(\theta_{\mu,t}) \\ &= \alpha_t^2\|x^\mu\|^4\|w(t)\|^2\sin^2(\theta_{\mu,t})\cos^2(\theta_{\mu,t})\end{aligned} \tag{14}$$

Therefore, inequality (13) reduces to:

$$\alpha_t\|x^\mu\|^2|\sin(2\theta_{\mu,t})| \leq 2\beta \tag{15}$$

Since $|\sin(2\theta_{\mu,t})| \leq 1$, if we manage to have $\alpha_t\|x^\mu\|^2 \leq 2\beta$ we will be safe. Therefore, in order to have small enough updates, it is sufficient to have:

$$\alpha_t \leq \frac{2\beta}{\max_\mu(\|x^\mu\|^2)} \tag{16}$$

Combining this equation with (12a), we notice that since $\lambda > 0$ and $\beta \ll 1$, we have $2\beta < 2 + \lambda_t$. Therefore, it is sufficient to choose α_t according to equation (16). However, equation (12b) will cause problems if $\frac{\lambda_t}{\|x^\mu\|^2\sin^2(\theta_{\mu,t})} > 2\beta$ which we have to address in our next reports.

Remark 4. ***A Remark about equation (12)****: In equation (12b), note that as $\theta_{\mu,t} \simeq 0^\circ$, we need large α's to ensure convergence. However, in such cases, $w(t)$ does not get updated at all because $\Delta w_t \simeq 0$. Therefore, one have to be careful to pick an initial $w(0)$ such that its projection on x^μ is not large, i.e. $\theta_{\mu,0} \gg 0^\circ$. In that case, if we show that $\theta_{\mu,t}$ increases monotonically, we can prove convergence. We can use induction on t to show that the required conitions are satisfied because at $t = 0$ we have $\lambda_t = 0$. Thus, we always can find an α ensuring $|y(\mu,1)| < |y(\mu,0)|$, which intuitively translates into $\theta_{\mu,1} > \theta_{\mu,0}$.*

Remark 5. ***Practical choice of*** *α: Even if we can find a proper α for each pattern in each iteration, one notices that for each μ (and t) we must use a different α to have $\frac{\lambda_t}{\|x^\mu\|^2\sin^2(\theta_{\mu,t})} < \alpha < \frac{2+\lambda_t}{\|x^\mu\|^2\sin^2(\theta_{\mu,t})}$. Therefore, it is possible that for some cases we can not have a unique α that works for all patterns. However, simulations show that a relatively small constant α is usually sufficient to ensure convergence.*

D. Making the Algorithm Parallel

Obviously, in order to find m constraints, we need to repeat algorithm 1 m 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 initial points, the algorithm converges to different distinct constraints most of the time. Besides that, as long as we have enough distinct constraints, the recall algorithm in the next section works just fine. Therefore, we will use the parallel version to have a faster algorithm in the end.

IV. Recall Phase: Expander Graphs

In the recall phase, we are given a noisy version of a pattern we have memorized before and asked to retrieve the correct pattern, i.e. eliminate the noise and return the memorized pattern. Let us denote the given pattern by $x^\mu = (x_1^\mu, \cdots, x_n^\mu)$, where the values x_j^μ belong to the alphabet S. Therefore, the initial states of the pattern nodes in Fig. 1 at the beginning of the recall phase is given by the vector $x = x^\mu + z$, where z is the noise with *integer* entries. We assume that the noise is integer valued and additive, and that the value of the noise added to the pattern is clipped to either 0 or $S-1$, when the sum of the noise and input is less than 0 or greater than $S-1$, respectively. Biologically speaking, the noise can be interpreted as a neuron skipping some spikes or firing more spikes than it should.

Since in the learning phase we have obtained the connectivity matrix of the neural graph in such a way that it is orthogonal to the memorized 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 $W \cdot (x^\mu + z) = W \cdot x^m y + W \cdot z = W \cdot z$.

In this section, we describe the recall algorithm for expander graphs as it gives us some intuition about how a neural network can benefit from its topological structure to eliminate noise while using simple iterative operations permitted by neurons. Later, we will extend this result to non-expander (but sparse) neural graphs.

A. The Recall Algorithm for Expander Graphs

We proposed a recall algorithm for expander sparse neural graphs in [2]. There, we assumed the connectivity matrix W is a expander sparse regular bipartite graph in which the degree of each pattern and constraint node is d_p and d_c, respectively. Furthermore, we assumed W_{ij} to represent the weight of the link between the j^{th} pattern node x_j and the i^{th} constraint node y_i. Hence d_c and d_p, as previously defined, denote the number of edges (degree) connected to the constraint and pattern neurons respectively.

The proposed algorithm comprises a series of forward and backward iterations. We suggested two different approaches that slightly differ from each other in the update rule at

Algorithm 2 Recall Algorithm: Winner-Take-All

Input: Connectivity matrix W, iteration $t_{\max}$

Output: $x_1, x_2, \ldots, x_n$

1: **for** $t = 1 \to t_{\max}$ **do**

2: *Forward iteration:* Calculate the weighted input sum $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each neuron y_i and set:

$$y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise}^2 \end{cases}.$$

3: *Backward iteration:* Each neuron x_j computes

$$g_j = \frac{\sum_{i=1}^{m} W_{ij} y_i}{d_p}.$$

4: Find

$$j^* = \arg\max_j |g_j|.$$

5: Update the state of winner j^*: If $g_{j^*} \neq 0$, then set $x_{j^*} = x_{j^*} + \text{sign}(g_{j^*})$.

6: $t \leftarrow t + 1$

7: **end for**

Algorithm 3 Recall Algorithm: Bit-Flipping

Input: Connectivity matrix W, threshold φ, iteration $t_{\max}$

Output: $x_1, x_2, \ldots, x_n$

1: **for** $t = 1 \to t_{\max}$ **do**

2: *Forward iteration:* Calculate the weighted input sum $h_i = \sum_{j=1}^{n} W_{ij} x_j$, for each neuron y_i and set:

$$y_i = \begin{cases} 1, & h_i < 0 \\ 0, & h_i = 0 \\ -1, & \text{otherwise}^2 \end{cases}.$$

3: *Backward iteration:* Each neuron x_j computes

$$g_j = \frac{\sum_{i=1}^{m} W_{ij} y_i}{\sum_{i=1}^{m} |d_p|}.$$

4: Update the state of each pattern neuron j according to $x_j = x_j + \text{sgn}(g_j)$ only if $|g_j| > \varphi$.

5: $t \leftarrow t + 1$

6: **end for**

pattern nodes. In the first one, given by algorithm 2, only the pattern node that receives the highest amount of feedback updates its state while the other pattern neurons maintain their current states. Therefore, it is a winner-take-all approach, which is well known and studied among the neuroscience community [7] and can be implemented by simple additional neural circuitry.

The second approach, given by algorithm 3, is less complex 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 remain intact otherwise. This algorithm is very similar to the *bit-flipping* algorithm of [23] for compressive sensing methods.

B. Number of Patterns and Distance Properties

In order to prove the subsequent results, we will need to employ *bipartite expander graphs* (as in [23]–[25]), which is defined in appendix A. We now show that there exist an exponential number of patterns x satisfying $Wx = 0$, when W is randomly chosen from the $(d_p, d_c, n, m)-$regular bipartite ensemble.

Theorem 1. *Let neurons have integer-valued states between 0 and $S-1$, and each row of the $m \times n$ matrix W contain exactly d_c ones (with the remaining entries being zero). Then, there exists a column vector b such that the system of linear equations $Wx = b$ has an exponential number of solutions.*

Proof: Since each component of the vector x can take any integer value from 0 to $S-1$, we have a total number of S^n such vectors. Suppose $y = Wx$. Then each component of y is between 0 and $d_c(S-1)$. Hence, there are at most $[d_c(S-1)+1]^m$ of such vectors y. Now if each component of the vector b can take any integer value from 0 to $S'-1$, then for $S' \geq d_c S$ there exists a vector b such that the system of equations $Hx = b$ has an integer solution and $S^n/[d_c(S-1)+1]^m \geq S^n/(d_c S)^m$ vectors of length n satisfy the set of equations. Therefore, for $m = n/2$ if $S > d_c$ we will have an exponential number of solutions for the system of equations. More generally, if $\log(S) > \frac{m}{n-m}\log(d_c)$ we can find a b such that we have an exponential number of solutions. ■

Next, we present a sufficient condition such that the minimum Hamming distance[2] between these exponential number of patterns is not too small.

Theorem 2. *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 obtained as a solution to $Wx = 0$, as before. 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 W_i denote the i^{th} column of W. If two patterns are at Hamming distance d from each other, then there exist non-zero integers $c_1, c_2, \ldots, c_d$ such that

$$c_1 W_{i_1} + c_2 W_{i_2} + \cdots + c_d W_{i_d} = 0, \tag{17}$$

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 by W, with $|\mathcal{P}| = d$. As in [23], we divide $\mathcal{N}(\mathcal{P})$ into two

[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.

[2]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.

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 (17) 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 $\lfloor \alpha n \rfloor + 1$. ■

C. 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 6. *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 $|Q| \leq 2$ 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 7. *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 3. *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 6 and 7. ■

D. 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 suboptimality 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 4. *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 $d_{p'}$ neighbours in common. As argued in the proof of Lemma 6, we have that

$$d_{p'} < 2d_p(1-\beta). \tag{18}$$

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 neighbours. If the noise corrupting these two pattern nodes 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 $\text{sign}(z_i) \neq \text{sign}(z_j)$. If $|z_i| = |z_j|$, then x_i (x_j) receives non-zero feedback only from the $d_p - d_{p'}$ edges in $\mathcal{N}(\{x_i\})\backslash Q$ (resp. $\mathcal{N}(\{x_j\})\backslash Q$) during the backward iteration, and the feedback is such that the noise is reduced at the end of the update. If on the other hand $|z_i| > |z_j|$, then x_i receives $-\text{sign}(z_i)$ along all d_p of it's incoming links during the backward iteration. However, x_j receives the correct $-\text{sign}(z_j)$ feedback along only the $d_p - d_{p'}$ edges from $\mathcal{N}(\{x_j\})\backslash Q$, and receives incorrect feedback of $\text{sign}(z_j)$ along the $d_{p'}$ edges from Q.

From the above and from (18), we conclude that when two pattern nodes are in error, at least one of the erroneous pattern nodes receives correct feedback along $d_p - 2d_p(1-\beta)$ or more edges; and, in the event of a node x_j (say) receiving incorrect feedback along some of it's incoming links, we have that $g_j = (-\text{sign}(z_j))\frac{d_p - 2d_{p'}}{d_p}$. For $\beta = \frac{4}{5}$, we have from (18) that $\text{sign}(g_j) = -\text{sign}(z_j)$ since $d_p - 2d_{p'} > 0$; hence irrespective of the value of γ, it is not possible that the node x_j is updated such that the noise magnitude is increased.

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 6, 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:

- At least one erroneous pattern node is updated in each iteration of our algorithm such that the noise magnitude is reduced.
- No erroneous pattern node can be updated such that the noise magnitude is increased.
- No pattern node other than the erroneous pattern nodes is updated.

■

E. 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 2, the choice of β depends on d_p, according to $\frac{1}{2} + \frac{1}{4d_p} < \beta < 1 - \frac{1}{d_p}$.
- Choose $d_c, S, S', r \triangleq m/n$ such that $S^n > (d_c S)^{rn}$ and $S' > d_c S$, so that Theorem 1 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.

V. Extension to Non-expander Graphs

So far ???

VI. Simultaneous Learn and Recall

So far, the algorithm yields a sparse matrix which satisfies the set of linear equations $Wx^\mu = 0$, for all vectors x^μ in the training data set. However, since we have to go over 2^k such vectors (and for each vector update m vectors of length n), the whole learning process is extremely slow even for rather small values of k. Therefore, instead of going over all possible patterns in the subspace, we pick a portion of them at random, perform the learning phase and when finished, go to directly the recall phase.

Now during the recall phase, two cases may occur: either we are given a noisy version of a pattern we have memorized before, or we will have a noisy version of a new pattern. One can distinguish these two cases based on the values returned by constraint nodes. In the former case, the deviation from the desired constraint values is rather small if the degree of corruption in the input pattern is not very high. Therefore, we can carry on with the usual error correcting algorithm in the recall phase.

In the latter case, where we are given a noisy version of a new pattern, the deviation from the constraints are usually high, indicating the existence of a pattern not learned before. In this case, we perform a learning phase. Note that since the pattern we are learning is usually noisy, we might have to repeat this process several times for a pattern to cancel out noise.

Obviously, there is a trade off between accuracy and speed here, but this scenario makes sense biologically as well. Because we do not learn everything at once but rather do the process in a gradual manner.

VII. Simulation Results

We have simulated the proposed learning algorithm in the multi-level architecture to investigate the block error rate of the suggested approach and the gain we obtain in error rates by adding a second level. We constructed 4 local networks, each with $n/4$ pattern and m constraint nodes.

A. Learning Phase

We have simulated the proposed learning algorithm over three different network sizes $n = 90, 300, 600$ with $m = 60, 100, 200$ constraints, respectively. To execute the learning phase, we first generate M patterns that belong to a subpace with dimension $k = n - m$ by multiplyig M *binary* vectors of length k to a sparse randomly generated $0/1$ matrix $G \in \mathcal{R}^{k\times n}$. We repeat the learning algorithm given by equation (4) until $y(\mu, t) \leq \epsilon$ for *all patterns* μ, where ϵ is a very small number. For avoiding numerical issues, we also normalized all the patterns. This does not affect the final result though. Furthermore, λ_t was bounded above so that it was always less than 0.5 to ensure that consecutive updates do not affect each other. Finally, in each iteration we put all entries less than 0.001 to zero manually.

Figure 2 shows the number of iterations executed before convergence is reached for different constraint nodes. We have investigated three different network sizes, i.e. $n = 90, 300, 600$. The corresponding number of constraints were $m = 60, 100, 200$, respectively.

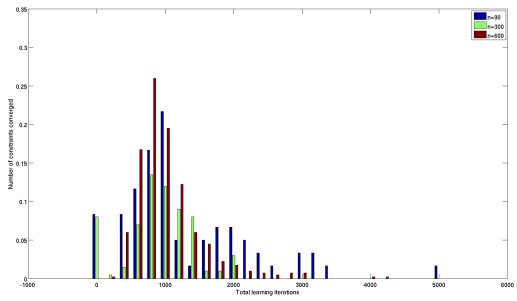

Fig. 2. Convergence rate for the specified number of constraints and different values of network sizes: $n = 90, 300, 600$. The corresponding number of constraints were $m = 60, 100, 200$, respectively

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.

Finally, figure 4 shows the MSE, given by the $\sum_{\mu=1}$: $My(\mu, t)$ in equation (4a), in each iteration for a sample constraint node for three different values of n. Overall, the MSE is overall decreasing. However, small increases happens due to the addition of the sparsity constraint. The amplitude of fluctuations (and in fact the convergence of the algorithm) depends heavily on the choice of parameters α and λ.

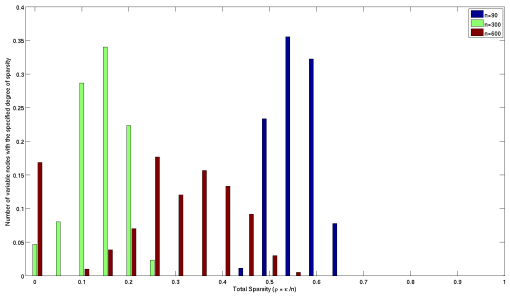

Fig. 3. The percentage of variable nodes with the specified sparsity measure and different values of network sizes: $n = 90, 300, 600$. The sparsity measure is defined as $\rho = \kappa/n$, where κ is the number of non-zero elements.

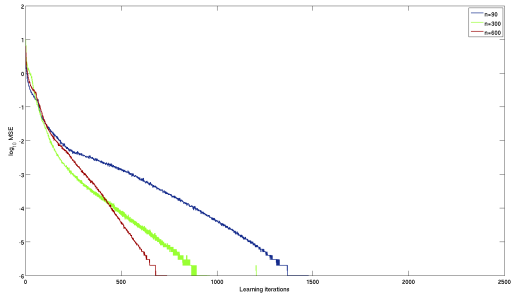

Fig. 4. MSE as a function of iteration for a random constraint node and three different values of n.

B. Recall Phase

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. As mentioned earlier, the errors are corrected first at the local and the at the global level. When finished, we compare the output of the first and the second level with the original (uncorrupted) pattern x. A pattern error is declared if the output does not match at each stage. Table VII-B shows the simulation parameters in the recall phase.

TABLE I
Simulation Parameters

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

Figure **??** illustrates the pattern error rates $n = 400$ with two different values of $k_t = 100$ and $k_t = 200$. The results are also compared to that of the bit-flipping algorithm in [2] to show the improved performance of the proposed algorithm. As

one can see, having a larger number of constraints at the global level, i.e. having a smaller k_t, will result in better pattern error rates at the end of the second stage. Furthermore, note that since we stop the learning after 99% of the patterns had learned, it is natural to see some recall errors even for 1 initial erroneous node.

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

TABLE II
GAIN IN PATTERN ERROR RATE (PER) FOR DIFFERENT VALUES OF $n = 400$ AND INITIAL NUMBER OF ERRORS

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

VIII. FUTURE WORKS

In order to extend the multi-level neural network, we must first find a way to generate patterns that belong to a subspace with dimensions $nL - m_g$, where m_g lies *within* the inside of bounds $L(n-k) < m_g < nL - k$. This will give us a way to investigate the trade off between the maximum number of memorizable patterns and the degree of error correction possible.

Furthermore, so far we have assumed that the second level enforces constraints in the same space. However, it is possible that the second level imposes a set of constraints in a totally different space. For this purpose, we need a mapping from one space into another. A good example is the written language. While they are local constraints on the spelling of the words, there are some constraints enforced by the grammar or the overall meaning of a sentence. The latter constraints are not on the space of letters but rather the space of grammar or meaning. Therefore, in order to for instance to correct an error in the word *_at*, we can replace _ with either W, to get *hat*, or c to get *cat*. Without any other clue, we can not find the correct answer. However, let's say say we have the sentence "*The _at ran away*". Then from the constraints in the space of meaning we know that the subject must be an animal or a person. Therefore, we can return *cat* as the correct answer. Finding a proper mapping is the subject of our future work.

ACKNOWLEDGMENT

The authors would like to thank Prof. Amin Shokrollahi for helpful comments and discussions. This work was supported by Grant 228021-ECCSciEng of the European Research Council.

APPENDIX A
EXPANDER GRAPHS

This section contains the definitions and the necessary background on expander 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*

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

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

The following result from [25] 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 B
EXPANDER GRAPHS

s

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