Multi-Level Error-Resilient Neural Networks with Learning

Amir Hesam Salavati and Amin Karbasi
School of Computer and Communication Sciences, Ecole Polytechnique Federale de Lausanne (EPFL)
Email: hesam.salavati@epfl.ch, amin.karbasi@epfl.ch,

***Abstract*—The problem of neural network association is to retrieve a previously memorized pattern from its noisy version using a network of neurons. An ideal neural network should include three components simultaneously: a learning algorithm, a large pattern retrieval capacity and resilience against noise. Prior works in this area usually improve one or two aspects at the cost of the third.**

Our work takes a step forward in closing this gap. More specifically, we show that by forcing natural constraints on the set of learning patterns, we can drastically improve the retrieval capacity of our neural network. Moreover, we devise a learning algorithm whose role is to learn those patterns satisfying the above mentioned constraints. Finally we show that our neural network can cope with a fair amount of noise.

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 in 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. In Section II we formally state the problem that is the focus of this work, namely neural association for a network of non-binary neurons. We then provide an overview of the related work in this area in Section II-A. We present our pattern learning algorithm in Section III and the multi-level network design in Section IV. The simulations supporting our analytical results are shown in Section VI. Finally future works are explained in Section VII.

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:

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

subject to

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

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

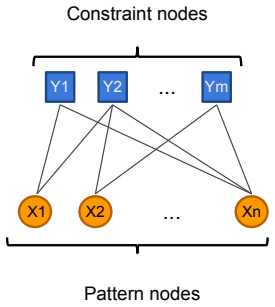

Fig. 1. A bipartite graph that represents the constraints on the training set.

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 $W \in \mathbb{R}^{m\times n}$ such that $Wx^\mu = 0$ for all $\mu = 1, \dots, C$.

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 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]. 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

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

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.

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 clique-based coding. In some cases, the proposed approach results in capacities of around $30n$, which is much larger than $O(n/\log(n))$ in other methods. In [12], the authors used low correlation sequences similar to those employed in CDMA communications to increase the storage capacity of Hopfield networks to n without requiring any separate decoding stage.

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 as well as the assumption that the weight matrix should be an expander. 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 focus on solving the above two problems in [2]. We start by proposing an iterative learning algorithm that identifies a *sparse* weight matrix W. The weight matrix W should satisfy a set of linear constraints $Wx^\mu = 0$ for all the patterns x^μ in the training data set, where $\mu = 1, \ldots, C$. We then propose a novel network architecture which eliminates the need for the expansion criteria while achieving better performance than the error correction algorithm proposed in [2].

Constructing a factor-graph model for neural associative memory has been also addressed in [22]. However, there, the authors propose a general message-passing algorithm to memorize any set of random patterns while we focus on memorizing patterns belonging to subspaces with sparsity in mind as well. The difference would again be apparent in the pattern retrieval capacity (linear vs. exponential in network size).

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 Algorithm

We are interested in an iterative algorithm that is simple enough to be implemented by a network of neurons. Therefore, we first relax (1) as follows:

$$\min \|\mathcal{X} \cdot w\|_2 - \lambda(\|w\|_2^2 - \epsilon) + \gamma(g(w) - q'). \qquad (2)$$

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. By calculating the derivative of the objective function and primal-dual optimization techniques we obtain the following iterative algorithm for (2):

Algorithm 1 Iterative Learning

Input: pattern matrix $\mathcal{X}$, stopping point p.
Output: w
  while $\|y(t)\|_{\max} > p$ **do**
    Compute $y(t) = \frac{\mathcal{X}.w(t)}{\|\mathcal{X}\|_2}$.
    Update $w(t+1) = \eta\left((1+2\lambda_t)w(t) - 2\alpha_t \frac{\mathcal{X}^\top y(t)}{\|\mathcal{X}\|_2}\right)_{\theta_t}$.
    Update $\lambda_{t+1} = \left[\lambda_t + \delta(\epsilon - \|w\|_2^2)\right]$.
    $t \leftarrow t+1$.
  end while

$$y(t) = \frac{\mathcal{X}\cdot w(t)}{\|\mathcal{X}\|_2} \tag{3a}$$

$$w(t+1) = (1+2\lambda_t)w(t) - 2\alpha_t \frac{\mathcal{X}^\top y(t)}{\|\mathcal{X}\|_2} - \gamma_t \nabla g(w) \tag{3b}$$

$$\lambda_{t+1} = \left[\lambda_t + \delta(\epsilon - \|w\|_2^2)\right] \tag{3c}$$

$$\gamma_{t+1} = \left[\gamma_t + \delta(g(w) - q')\right] \tag{3d}$$

where t denotes the iteration number, $\mathcal{X}^\top$ is the transpose of matrix $\mathcal{X}$, δ and α_t are small step sizes and $[\cdot]_+$ denotes $\max(\cdot, 0)$.

For our choice of $g(w)$, the i^{th} entry of the function $f(w) = \nabla g(w)$, denoted by $f_i(w)$ reduces to $2\sigma w_i(1-\tanh(\sigma w_i^2)^2)$. For very small values of w_i, $f_i(w) \simeq w_i$ and for large values of w_i, $f_i(w) \simeq 0$. Therefore, by looking at (3b) we see that the last term is pushing small values in $w(t+1)$ towards zero while leaving the larger values intact. Therefore, we remove the last term completely and enforce small entries to zero in each update which in turn enforces sparsity. The final iterative learning procedure is shown in Algorithm 1.

Here, θ_t is a positive threshold at iteration t and $\eta(.)_{\theta_t}$ is the *point-wise* soft-thresholding function given below:

$$\eta(u)_\theta = \begin{cases} u & \text{if } u > \theta, \\ u & \text{if } u < -\theta, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Remark 2. *the above choice of soft-theresholding function is very similar to the one selected by Donoho et al. in [1] in order to recover a sparse signal from a set of measurements. The authors prove that their choice of soft-threshold function results in* optimal *sparsity-undersampling trade-off.*

The next theorem derives the necessary conditions on α_t, λ_t and θ_t such that Algorithm 1 converges to a sparse solution.

Theorem 1. *If $\theta_t \to 0$ as $t \to \infty$ and if λ_t is bounded above by $a_{\min}/(a_{\max} - a_{\min})$, then there is a proper choice of α_t in every iteration t that ensures constant decrease in the objective function $\|\mathcal{X}.w(t)\|_{\max}$. Here $a_{\min} = \min_\mu \|x^\mu\|^2/\|\mathcal{X}\|^2$ and $a_{\max} = \max_\mu \|x^\mu\|^2/\|\mathcal{X}\|^2$. For $\lambda_t = 0$, i.e. $\|w(t)\|_2 \geq \epsilon$, picking $0 < \alpha_t < 1$ ensures gradual convergence.*

Sketch of the proof: Let $E(t) = \|y(t)\|_{\max}$. We would like Let $E(t) = \|y(t)\|_{\max}$. We would like to show that $E(t+1) < E(t)$ for all iterations t. To this end, let us denote $(1+2\lambda_t)w(t) - 2\alpha_t \frac{\mathcal{X}^T y}{\|\mathcal{X}\|_2}$ by $w'(t)$. Furthermore, let the function $\chi(u;\theta_t)$ be $u - \eta(u)_{\theta_t}$. Rewriting the second step of algorithm (1) we will have:

$$w(t+1) = w'(t) - \chi(w'(t);\theta_t) \tag{5}$$

Now we have

$$\begin{aligned} E(t+1) &= \|y(t+1)\|_{\max} = \|\frac{\mathcal{X}.w(t+1)}{\|\mathcal{X}\|_2}\|_{\max} \\ &\leq \|\frac{\mathcal{X}.w'(t)}{\|\mathcal{X}\|_2}\|_{\max} + \|\frac{\mathcal{X}.\chi(w'(t);\theta_t)}{\|\mathcal{X}\|_2}\|_{\max} \\ &\leq \|\frac{\mathcal{X}.w'(t)}{\|\mathcal{X}\|_2}\|_{\max} + \theta_t \frac{\|\mathcal{X}\|_{\max}}{\|\mathcal{X}\|_2} \\ &\leq \|\frac{\mathcal{X}.w'(t)}{\|\mathcal{X}\|_2}\|_{\max} + \theta_t \end{aligned} \tag{6}$$

where the last inequality follows because $\|\mathcal{X}\|_{\max} \leq \|\mathcal{X}\|_2$. Now expanding $\frac{\mathcal{X}.w'(t)}{\|\mathcal{X}\|}$ we will get

$$\begin{aligned} \frac{\mathcal{X}.w'(t)}{\|\mathcal{X}\|_2} &= (1+2\lambda_t)y(t) - 2\alpha_t \frac{\mathcal{X}\mathcal{X}^T y}{\|\mathcal{X}\|_2^2} \\ &= \left[(1+2\lambda_t)I_{C\times C} - 2\alpha_t \frac{\mathcal{X}\mathcal{X}^T}{\|\mathcal{X}\|_2^2}\right] y(t) \end{aligned} \tag{7}$$

Denoting the matrix $(1+2\lambda_t)I_{C\times C} - 2\alpha_t \frac{\mathcal{X}\mathcal{X}^T}{\|\mathcal{X}\|_2^2}$ by D_t, we can further simplify inequality (6):

$$\begin{aligned} E(t+1) &\leq \|D_t y(t)\|_{\max} + \theta_t \\ &\leq \|D_t\|_{\max}\|y(t)\|_{\max} + \theta_t \\ &= \|D_t\|_{\max} E(t) + \theta_t \end{aligned} \tag{8}$$

Where $D(t) = (1+2\lambda_t)I_{C\times C} - 2\alpha_t \frac{\mathcal{X}\mathcal{X}^T}{\|\mathcal{X}\|_2^2}$. Therefore, if we set $\theta_t = 1/t$ (i.e. $\theta_t \to 0$ as $t \to \infty$) and ensuring $\|D_t\|_{\max} < 1$ we get $E(t+1) < E(t)$. The second requirement requires that $|D_{ij}(t)| < 1$ for all elements (i,j) of D_t. Therefore, by letting $A = XX^T$ we must have the following relationship for diagonal elements:

$$|1 + 2\lambda_t - 2\alpha_t \frac{A_{ii}}{\|A\|_2}| < 1 \tag{9}$$

which yields

$$\lambda_t < \alpha_t \frac{A_{ii}}{\|A\|_2} < 1 + \lambda_t, \ \forall i = 1, \dots C$$

Since $\lambda_t \geq 0$ for all t and $0 < A_{ii} \leq \|A\|_2$, the right hand side of the above inequality is satisfied if $\alpha_t < 1$. The left-hand side is satisfied for $\alpha_t > \lambda_t/a_{\min}$, where $a_{\min} = \min_i A_{ii}/\|A\|_2$. Therefore, if $\lambda_t \leq a_{\min}/(a_{\max} - a_{\min})$ there exists and α_t ensuring $\|D\|_{max} < 1$. If $\lambda_t = 0$, this is simply equivalent to having $0 < \alpha < 1$.

■

IV. Multi-level Network Architecture

In the previous section, we discussed the details of a simple iterative learning algorithm which yields rather sparse graphs. Now in the recall phase, we propose a network structure together with a simple error correction algorithm (similar to

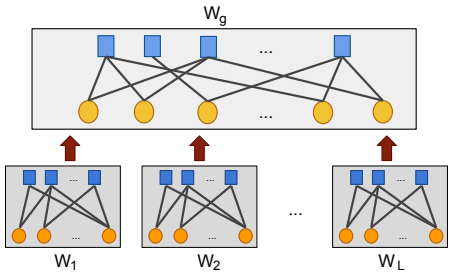

Fig. 2. A two-level error correcting neural network.

Algorithm 2 Error Correction

Input: pattern matrix $\mathcal{X}$, 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

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

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**

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.

the one in [2]) to achieve good block error rates in response to noisy input patterns. The suggested network architecture is shown in Figure 2. To make the description clear and simple we only concentrate on a two-level neural network. However, the generalization of this idea is trivial and left to the reader.

The proposed approach is in contrast to the one in suggested in [2] where the authors exploit a single-level neural network with a sparse and *expander* connectivity graph to correct at least two initial input errors. However, enforcing expansion on connectivity graphs in a gradual neural learning algorithm is extremely difficult, specially when the algorithm is required to be very simple Therefore, we use the learning algorithm explained above, which yields a *rather* sparse and not necessarily expander graph, and improve the error correction capabilities by modifying the network structure and error correcting algorithm.

The idea behind this new architecture is that we divide the input pattern of size n into L sub-patterns of length n/L. Now we feed each sub-pattern to a neural network which enforces m constraints[3] on the sub-pattern in order to correct the input errors. The local networks in the first level and the global network in the second level use Algorithm 2, which is a variant of the "bit-flipping" method proposed in [2], to correct the errors. Intuitively, if the states of the pattern neurons x_i correspond to a pattern from $\mathcal{X}$ (i.e., the noise-free case), then for all $i = 1, \ldots, m$ we have $y_i = 0$. The quantity g_j can be interpreted as feedback to pattern neuron x_j from the constraint neurons. Hence, 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.

Theorem 2. *Algorithm 2 can correct a single error in the input pattern with high probability if φ is chosen large enough.*

Proof: In the case of a single error, we are sure that the corrupted node will always be updated towards the correct direction. For simplicity, let's assume the first pattern neuron is the noisy one. Furthermore, let $z = \{1, \ldots, 0\}$ be the noise vector. Denoting the i^{th} column of the weight matrix by W_i, we will have $y = \text{sign}(W_1)$. Then in algorithm 2 $g_1 = 1 > \varphi$. This means that the noisy node gets updated towards the correct direction.

Therefore, the only source of error would be a correct node gets updated mistakenly. Let P_{x_i} denote the probability that a correct pattern neuron x_i gets updated. This happens if $|g_{x_i}| > \varphi$. For $\varphi = 1$, this is equivalent to having $W_i \cdot \text{sign}(z_1 W_1) = \|W_i\|_0$. Note that $W_i \cdot \text{sign}(W_1) < \|W_i\|_0$ in cases that the neighborhood of x_i is different from the neighborhood of x_1 among the constraint nodes. More specifically, in the case that $\mathcal{N}(x_i) \neq \mathcal{N}(x_1)$, there are non-zero entries in W_i while W_1 is zero and vice-versa. Therefore, letting P'_{x_i} being the probability of $\mathcal{N}(x_i) \neq \mathcal{N}(x_1)$, we note that

$$P_{x_i} \leq P'_{x_i}$$

Therefore, to get an upper bound on P_{x_i}, we bound P'_{x_i}.

Let λ_i be the fraction of pattern neurons with degree i, $\bar{d} = \sum_i \lambda_i d_i$ be the average degree of pattern neurons and finally d_{min} be the minimum degree of pattern neurons. Then, we know that a noisy pattern neuron is connected to $\bar{d}$ constraint neurons on average. Therefore, the probability of x_i and x_1 share exactly the same neighborhood would be:

$$P'_{x_i} = \left(\frac{\bar{d}}{m}\right)^{d_{x_i}} \tag{10}$$

Taking the average over the pattern neurons, we have

$$\begin{aligned} P'_e &= \Pr\{x \in \mathcal{C}_t\} \mathbb{E}_{d_{x_i}}\{P'_{x_i}\} \\ &= (1 - \frac{1}{n}) \lambda(\frac{\bar{d}}{m}) \\ &= \lambda(\frac{\bar{d}}{m}) \end{aligned} \tag{11}$$

[2]Note that although we do not allow neurons to have negative outputs, the set of outputs $\{-1, 0, 1\}$ can be easily implemented by sending $\{0, 1, 2\}$ and shift the response in the pattern neurons in each iteration. The shifting can be done by modifying the firing threshold for each neuron.

[3]The number of constraints for different networks can vary. For simplicity of notifications we assume equal sizes.

where $\mathcal{C}_t$ is the set of correct nodes at iteration t and $\lambda(x) = \sum_i \lambda_i x^i$.

Therefore, the probability of correcting one noisy input, $P_c = 1 - P_e \geq 1 - P'_e$ would be

$$\begin{aligned} P_c &\geq 1 - \lambda(\frac{\bar{d}}{m}) \\ &\geq 1 - \left(\frac{\bar{d}}{m}\right)^{d_{min}} \end{aligned} \tag{12}$$

■

Given that each local network is able to correct one pattern, L such networks can correct L input errors *if* they are separated such that only one error appears in the input of each local network. Otherwise, there would be a probability that the network could not handle the errors. In that case, we feed the overall pattern of length n to the second layer with the connectivity matrix W_g, which enforces m_g *global* constraints. And since the probability of correcting two erroneous nodes increases with the input size, we expect to have a better error correction probability in the second layer. Therefore, using this simple scheme we expect to gain a lot in correcting errors in the patterns. In the next section, we provide simulation results which confirm our expectations and show that the block error rate can be improved by a factor of 100 in some cases.

A. Some remarks

First of all, one should note that the above method only works if there is some redundancy at the global level as well. If the set of weight matrices $W_1, \ldots, W_L$ define completely separate sub-spaces in the n/L-dimensional space, then for sure we gain nothing using this method.

Secondly, there is no need to have the dimension of the subspaces to be equal to each other. We can have different lengths for the sub-patterns belonging to each subspace and different number of constraints for that particular sub-space. This gives us more degree of freedom as well since we can spend some time to find the optimal length of each sub-pattern for a particular training data set.

Thirdly, the number of constraints for the second layer affects the gain one obtains in the error performance. Intuitively, if the number of global constraints is large, we are enforcing more constraints so we expect obtaining a better error performance. We can think of determining the number even adaptively, i.e. if the error performance that we are getting is unacceptable, we can look deeper in patterns to identify their internal structure by searching for more constraints. This would be a subject of our future research.

V. Pattern Retrieval Capacity

the following theorem will prove that the proposed neural architecture is capable of memorizing an exponential number of patterns.

Theorem 3. *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 $k_g = rn$ for some $0 < r < 1$. Then, there exists a set of such vectors for which $C = a^{rn}$, with $a > 1$, and rank($\mathcal{X}$) $= k_g < n$, and such that they can be memorized by the neural network given in figure 2.*

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

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

We start constructing the patterns in the data set as follows: consider a random vector $u^\mu \in \mathbb{R}^{k_g}$ with integer-valued-entries between 0 and $\upsilon - 1$, where $\upsilon \geq 2$. We set the pattern $x^\mu \in \mathcal{X}$ to be $x^\mu = u^\mu \cdot G$, *if* all the entries of x^μ are between 0 and $S-1$. 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 $x^\mu_j = u^\mu \cdot \eth_j$, where $\eth_j$ is the j^{th} column of G. Suppose $\eth_j$ has d_j non-zero elements. Then, we have:

$$x^\mu_j = u^\mu \cdot \eth_j \leq d_j(\gamma - 1)(\upsilon - 1)$$

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

$$S - 1 \geq d^*(\gamma - 1)(\upsilon - 1) \tag{13}$$

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

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

VI. 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 generated a sample data set of $C = 10000$ patterns of length n where each block of $n/4$ belonged to a subspace of dimension $k < n/4$. Note that C can be an exponential number in n. However, we selected $C = 10000$ as an example to show the performance of the algorithm because even for small values of k, and exponential number in k will become too large to handle numerically. The result of the learning algorithm is four different local connectivity matrices $W_1, \ldots, W_4$ as well as a global weight matrix W_g. The number of local constraints was $m = n/4 - k$ and the number of global constraints was $m_g = n - k_g$, where k_g is dimension of the subspace for

overall pattern. The learning steps are done until 99% of the patterns in the training set converged. Table VI-A summarizes other simulation parameters. For cases where $\lambda_t = 0$, α_t was

TABLE I
SIMULATION PARAMETERS

Parameter	δ	θ_t	α_t (when $\lambda_t \neq 0$)	ϵ	p
Value	10	$0.25/t$	$\min(\frac{\lambda_t}{a_{\min}}, 1+\lambda_t)$	0.01	$\frac{0.01}{\|X\|_2}$

fixed to 0.49.

Table VI-A shows the average number of iterations executed before convergence is reached for different constraint nodes at the local and global level. It also gives the average sparisty ratio for the columns of matrix W. The sparsity ratio is defined as $\rho = \kappa/n$, where κ is the number of non-zero elements. From the figure one notices that as n increases, the vectors become sparser.

TABLE II
AVERAGE NUMBER OF CONVERGENCE ITERATIONS AND SPARSITY IN THE LOCAL AND GLOBAL NETWORKS FOR $n = 400$

	Sparsity Ratio		Convergence Rate	
	$k_g = 100$	$k_g = 200$	$k_g = 100$	$k_g = 200$
Local	0.28	0.32	4808	5064
Global	0.22	0.26	14426	33206

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 VI-B shows the simulation parameters in the recall phase.

TABLE III
SIMULATION PARAMETERS

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

Figure 3 illustrates the pattern error rates $n = 400$ with two different values of $k_g = 100$ and $k_g = 200$. 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_g, will result in better pattern error rates at the end of the second stage. Furthermore, note that since we stop the learning after 99% of the patterns had learned, it is natural to see some recall errors even for 1 initial erroneous node.

Table VI-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.

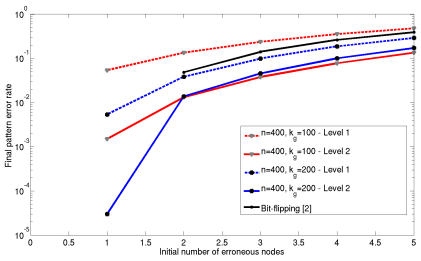

Fig. 3. Pattern error rate against the initial number of erroneous nodes

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

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

C. Comparison with Previous Work

For the sake of completeness, table V compares the proposed algorithm with the previous work from three different perspectives: the pattern retrieval capacity, the number of initial errors that can be corrected in the recall phase (denoted by e), the existence of an online iterative learning algorithm, and if there are any other restrictions such as the focus of the algorithm on particular patterns with some redundancy. In all cases it is assumed that the length of patterns is n.

TABLE V
NEURAL ASSOCIATIVE MEMORIES COMPARED TOGETHER FOR A PATTERN OF SIZE n

Algorithm	C	e	Learning?	Restrictions?
[4]	$O(n/\log(n))$	$O(n)$	yes	no
[13]	$n/2$	> 1	no	no
[14]	$0.15n$	> 1	no	no
[10]	n	> 1	no	no
[18]	$> n^a$	> 1	yes	PWR[b]
[17]	$O\left(n^{p-2}\right)$ [c]	> 1	yes	no
[2]	$O(a^n)^b$	> 2	no	PWR[d], EG[e]
This paper	$O(a^n)^b$	> 1	yes	PWR[d]

[a] The authors do not provide exact relationship for the pattern retrieval capacity. However, they show that for a particular setup with $n = 2048$, we have $C = 60000$.
[b] PWR stands for Patterns With Redundancy.
[c] p is the order of correlations considered among patterns.
[d] $a > 1$ is determined according to network parameters.
[e] EG stands for Expander Graphs.

VII. 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 h, 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.

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 Int. Theory Work., 2011, pp. 80-84.

[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*, IEEE Intl. Symp. Inform. Theory (ISIT-11), 2011, pp. 850-854.

[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. 1087-1096.

[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] A. Braunstein, R. Zecchina, *Learning by message-passing in networks of discrete synapses*, Phys. Rev. Lett., Vol. 96, No. 3, 2006, pp. 030201-1-030201-4

[23] D. J. Amit, S. Fusi, *Learning in neural networks with material synapses*, Neur. Comp., Vol. 6, No. 5, 1994, pp. 957-982.