Iterative Learning in Modular Associative Memories

Anonymous Author(s)
Affiliation
Address
email

Abstract

The task of a neural associative memory is to retrieve a set of previously memorized patterns from their noisy versions by using a network of neurons. Hence, an ideal network should be able to 1) gradually learn a set of patterns, 2) retrieve the correct pattern from noisy queries and 3) maximize the number of memorized patterns while maintaining the reliability in responding to queries. We show that by considering the inherent redundancy in the memorized patterns, one can obtain all the mentioned properties at once. This is in sharp contrast with the previous work that could only improve one or two aspects at the expense of the third. More specifically, we devise an iterative algorithm that learns the redundancy among the patterns. The resulting network has a retrieval capacity that is exponential in the size of the network. Lastly, by considering the local structures of the network, the asymptotic error correction performance can be made linear in the size of the network.

1 Introduction

The ability to memorize a large set of patterns and reliably retrieve them in the presence of noise, are among the main reasons that attracted a large body of research to neural networks for the past three decades. This problem, so called "associative memory", is in spirit very similar to reliable information transmission faced in communication systems where the goal is to efficiently decode a set of transmitted patterns over a noisy channel.

Despite this similarity and common methods deployed in both fields (graphical models, iterative algorithms, to name a few), we have witnessed a huge gap between the efficiency achieved by them. More specifically, by deploying modern coding techniques, it was shown that the number of reliably transmitted patterns over a noisy channel can be made *exponential* in n, the length of the patterns. This was achieved by intelligently imposing redundancy among transmitted patterns. In contrast, the maximum number of patterns that can be reliably memorized by the current neural networks scales *linearly* in the size of the network. This is due to the 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 (see, for example, [1], [2], [3], [4]).

To increase the storage capacity of neural networks beyond the current linear scaling Kumar et al. [5] suggested a new formulation of the problem where only a suitable set of patterns was considered for storing. To enforce the set of constraints, they formed a bipartite graph (as opposed to a complete graph considered in the earlier work) where one layer feeds the patterns to the network and the other takes into account the inherent structure. The role of bipartite graph is indeed similar to the Tanner graphs used in modern coding techniques. Under the conditions that the bipartite graph is given and it is an expander, they provided an algorithm that is guaranteed to correct a single error.

In this work, we follow the same line of thought. More precisely, we aim at memorizing a random subset of patterns of length n drawn from a training set of dimension $k < n$. Clearly, this set of

patterns enjoys a certain degree of *redundancy*, i.e., they come from a lower dimensional space. A natural way to enforce this structure on patterns is to find the connectivity matrix of the bipartite graph which is orthogonal to all of the patterns. By making use of this redundancy, we can increase the number of patterns that can be memorized from linear to exponential. Moreover, this structure leaves us with enough redundancy among patterns that we can potentially increase the error correction capability of our neural network from constant to linear.

Apart from efficiency, we also devise an iterative algorithm that learns the connectivity matrix associated with the bipartite graph. Our algorithm reweighs the edges of the bipartite graph once a new set of patterns are presented. As a result, the learning phase of our neural network needs not wait to see all the patterns at once before starting to learn them.

The remainder of this paper is organized as follows. We first review briefly the related work. We then proceed to Sections 3 and 4 to state our notation and describe our model. Our main results are outlined in Sections 5-7. Finally, we compare our method with the previous art in Section 8.

2 Related Work

Arguably, the Hopfield network is the first auto-associative neural mechanism capable of learning a set of patterns and recalling them later. By utilizing the same learning rules as Hebb's [6], Hopfield considered a neural network of size n with binary state neurons. It was shown by McEliece et al. that the capacity of a Hopfield network under vanishing block error is bounded by $\mathcal{C} = (n/2\log(n))$ [7].

With the hope of increasing the capacity of the Hopfield network, an extension of associative memories to non-binary states has also been explored in the past. In particular, Jankowski et al. [3] investigated a multi-state complex-valued neural associative memory where each neuron can be assigned a multivalued state from the set of complex numbers. It was shown by Muezzinoglu et al. [4] that the capacity of such networks can be increased to $\mathcal{C} = n$ at the cost of a prohibitive weight computation mechanism.

Recently, in order to increase the capacity and robustness, a line of work considered exploiting the inherent structure of the patterns. This is done by either making use of the correlations among the pattern or memorizing only those patterns that have some sort of redundancy. Note that they differ from the previous methods in one important aspect: not any possible set of patterns is considered for learning. By utilizing Walsh-Hadamard sequences, Berrou and Gripon [8] were among the first who demonstrated that considerable improvements in the pattern retrieval capacity of Hopfield networks is possible, albeit still not passing the linear boundary on the capacity.

By deploying higher order neural models, in contrast to the pairwise correlation considered in Hopfield networks, Peretto et al. [9] showed that the storage capacity can be improved to $\mathcal{C} = O(n^{p-2})$, where p is the degree of correlation. In such models, the state of the neurons not only depends on the state of their neighbors, but also on the correlations among them. Again, the main drawback lies in the prohibitive computational complexity required by the learning phase. To address this difficulty, while being able to capture higher-order correlations, a new model based on bipartite graphs was introduced in [5], and was further explored in [12]. Under the restrictive assumptions that the bipartite graph is fully known, sparse, and expander, their proposed algorithm increased the pattern retrieval capacity to $\mathcal{C} = O(a^n)$, for some $a > 1$. The lack of a learning algorithm as well as the expander assumption are among the reasons that reduces the capabilities of this model. To improve upon [5], authors in [12] proposed a multi-level neural network and provided evidence that such a network presents better error correction performance.

In this paper, we adopt the same approach as in [5]. We first devise an iterative learning algorithm that finds a *sparse* weight matrix W satisfying a set of linear constraints $W \cdot x = 0$ for all the patterns x drawn from the training data set $\mathcal{X}$. We show that the capacity of the resulting network is exponential in the size of the network. We then propose a novel error correction algorithm that corrects a linear fraction of errors in the recall phase. In contrast to [12], we consider overlapping clusters. We show theoretically and through exhaustive simulations that introducing overlaps among clusters improves the error correction performance significantly.

We should finally mention that learning linear constraints by a neural network is hardly a new topic. It is shown by [10] and [11] that one can learn a matrix orthogonal to a set of patterns in the training

set by using simple neural learning rules. However, finding such a matrix subject to the sparsity constraints has not been addressed before. This problem can also be regarded as an instance of compressed sensing [13]. With few exceptions, including [14] and [15], the decoders proposed in the compressed sensing area are far too complicated to be implementable by a neural network only capable of performing simple operations.

3 Notation and Definitions

Let $[q]$ denote the set of first q integers, namely, $[q] = \{1, 2, \cdots, q\}$.
Neurons: In our model, a neuron can assume an integer-valued state from the set $\mathcal{S} = \{0, \ldots, S-1\}$. A neuron updates its state based on the states of its neighbour $\{s_i\}_{i=1}^{n}$ as follows. It first computes a weighted sum $h = \sum_{i=1}^{n} w_i s_i$, where w_i denotes the weight of the input link from s_i. It then updates its state by passing the value of h through a non-linear function $f : \mathbb{R} \to \mathcal{S}$.

Clusters: In our model, a neural associative memory is represented by a bipartite graph G. It has n pattern nodes, $x_1, x_2, \ldots, x_n$, and m constraint nodes, $y_1, y_2, \ldots, y_m$. The graph G itself is composed of L clusters $G^{(1)}, G^{(2)}, \ldots, G^{(L)}$ each of which is again a bipartite graph. More specifically, cluster $G^{(i)}$ consists of n_i pattern nodes $x_1^{(i)}, x_2^{(i)}, \ldots, x_{n_i}^{(i)}$, and m_i constraint nodes $y_1^{(i)}, y_2^{(i)}, \ldots, y_{m_i}^{(i)}$.

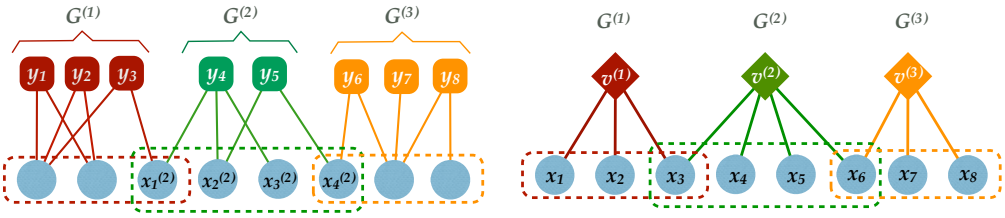
Degree Distributions: For the bipartite graph G with n pattern nodes, let Λ_i denote the number of variable nodes of degree i, so that $\sum_i \Lambda_i = n$. Likewise, denote the number of constraint nodes of degree i by R_i, so that $\sum_i R_i = m$. Since the edge counts must match up, we have $\sum_i i\Lambda_i = \sum_i iR_i$. It is convenient to introduce the compact notation $\Lambda(z) = \sum_i \Lambda_i z^i$ and $R(z) = \sum_i R_i z^i$. We call Λ and R the pattern and constraint degree distributions from a *node perspective*. In the same fashion, we can define degree distributions for clusters $G^{(1)}, G^{(2)}, \ldots, G^{(L)}$. More precisely, we denote by $\Lambda^{(i)}$ and $R^{(i)}$ the pattern and constraint degree distributions of cluster $G^{(i)}$. The degree distributions Λ and R (resp., $\Lambda^{(i)}$ and $R^{(i)}$) define an ensemble of randomly generated bipartite graphs. To analyse the performance of our denoising algorithm, we assume that the bipartite graph G (resp., $G^{(i)}$) is sampled randomly from this ensemble (see [19] for more details). In cluster $G^{(i)}$, we denote by $d_{\text{avg}}^{(i)}$ and $d_{\min}^{(i)}$ the average and minimum degrees of the pattern nodes, respectively. For the asymptotic analysis carried out in our work, it is more convenient to define the degree distributions from an *edge perspective*. To this end, we define $\lambda(z) = \sum_j \lambda_j z^{j-1}$ and $\rho(z) = \sum_j \rho_j z^{j-1}$. It is easy to see that λ_j (resp., ρ_j) is equal to the fraction of edges that connect to pattern (resp., constraint) nodes of degree j. As earlier, we denote by $\lambda^{(i)}$ and $\rho^{(i)}$ the pattern and constraint degree distributions of cluster $G^{(i)}$ from the edge perspective.

4 Problem Statement

In neural associative memories - the subject of this work - the goal is to design a neural network capable of memorizing a large set of patterns $\mathcal{C}$ from a data set $\mathcal{X}$ (learning phase), and recalling them later in presence of noise (recall phase).

Learning phase: Each pattern $x = (x_1, x_2, \ldots, x_n)$ is a vector of length n, where $x_i \in \mathcal{S}$ for $i \in [n]$. In this work, our focus is on memorizing the patterns with strong *local correlation* among the entries. More specifically, we divide the entries of each pattern x into L *overlapping* sub-patterns of lengths $n_1, \ldots, n_L$, so that $\sum n_i \geq n$. Note that due to overlaps, a pattern node can be a member of multiple sub-patterns, as shown in Figure 1a. We denote the i-th sub-pattern by $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_{n_i}^{(i)})$. To enforce local correlations, we assume that the sub-patterns $x^{(i)}$ form a subspace of dimension $k_i < n_i$. This is done by impose linear constraints on each cluster. These linear constraints are captured in the form of dual vectors as follows. Let $\{w_1^{(i)}, w_2^{(i)}, \ldots, w_{m_i}^{(i)}\}$ be a set of dual vectors orthogonal to all the sub-patterns $x^{(i)}$ of cluster i. In the learning phase, we would like to memorize these patterns by finding a set of non-zero vectors $w_1^{(i)}, w_2^{(i)}, \ldots, w_{m_i}^{(i)}$ that are orthogonal to the set of sub-patterns $x^{(i)}$, i.e.,

$$y_j^{(i)} = (w_j^{(i)})^T \cdot x^{(i)} = 0, \quad \forall j \in [m_i] \,\&\, \forall i \in [L]. \tag{1}$$



(a) Bipartite graph G. (b) Contraction graph $\widetilde{G}$.

Figure 1: The proposed neural associative memory with overlapping clusters.

The weight matrix $W^{(i)} = [w_1^{(i)}|w_2^{(i)}|\dots|w_{m_i}^{(i)}]^T$ of cluster i is created by putting all the dual vectors next to each other. Equation (1) can be written equivalently as $W^{(i)} \cdot x^{(i)} = 0$. Therefore, the goal is to devise a low complexity algorithm that updates the weights of $W^{(i)}$'s (for $i \in [L]$) once it encounters a new pattern $y \in \mathcal{X}$. We should stress here that contrary to our iterative algorithm, many proposed learning algorithms in the literature can learn a set of patterns only after the whole set is presented to the neural network. More importantly, they are usually unable to learn a new set of patterns. Removing this restriction is one of the major contributions of our work.

One can easily map the local constraints imposed by $W^{(i)}$'s into a global constrain by introducing a global weight matrix W of size $m \times n$. The first m_1 rows of the matrix W correspond to the constraints in the first cluster, the rows m_1+1 to m_1+m_2 correspond to the constraints in the second cluster, and so forth. Hence, by inserting the zero entries at proper positions, we can construct the global constraint matrix W. We will use both the local and global connectivity matrices to eliminate noise in the recall phase.

Recall phase: Once the set of patterns $\mathcal{X}$ have been memorized, a desirable neural network should be able to retrieve an already memorized pattern from its corrupted version (of course, if the intensity of noise is not moderate, there is no hope for recovery). Needless to say that the recall process should be implementable by simple operations mentioned earlier.

In the recall, phase a noisy version, say y, of an already learned pattern $x \in \mathcal{X}$ is given. Here, we assume that the noise is an additive vector of size n, denoted by e, whose entries assume values independently from $\{-1, 0, +1\}$ with corresponding probabilities $p_{-1} = p_{+1} = p_e/2$ and $p_0 = 1 - p_e$. To ensure that the amount of noise is moderate, we also need to assume that the error probability, p_e, is less than p_0. As before, we denote by $e^{(i)}$, the realization of noise on the sub-pattern $x^{(i)}$. In formula, $y = x+e$ (modulo S). Note that $W \cdot y = W \cdot e$ and $W^{(i)} \cdot y^{(i)} = W^{(i)} \cdot e^{(i)}$. Therefore, the goal will be to remove the noise e and obtain the desired pattern x as the true states of the pattern neurons. This task will be accomplished by exploiting the fact that we have chosen the set of patterns $\mathcal{X}$ to satisfy the set of constraints $W^{(i)} \cdot x^{(i)} = 0$.

Capacity: The last issue we look at in this work is the retrieval capacity $\mathcal{C}$ of our proposed method. Formally, the retrieval capacity is defined in terms of the maximum number of patterns that a neural network can learn and distinguish later. By construction, we show that the retrieval capacity of our network is exponential in the size of the network.

5 The Learning Algorithm

In this section, we discuss the learning algorithm for a single cluster ℓ. The case of multiple clusters would be a straightforward extension of the suggested algorithm. Since the patterns lie in a subspace of dimension n_ℓ, we adapt the algorithm proposed in [16] and [10] to learn the null-space basis of the subspace defined by the patterns. Due to requirements of the denoising algorithm used in the recall phase, we need one more property: the basis vectors should be sparse. To this end, we add an additional term to penalize non-sparse solutions during the learning phase. Furthermore, we are not looking for an orthogonal basis as in [10]. Instead, we would like to find m vectors (from the dual space of $\mathcal{X}$) that are orthogonal to the patterns.

$$\min_{w} \sum_{x \in \mathcal{X}} |x \cdot w|^2 + \beta g(w), \quad \text{s.t. } \|w\|_2 = 1. \tag{2}$$

Algorithm 1 Iterative Learning

Input: Set of patterns $\mathcal{X}$ with $|\mathcal{X}| = \mathcal{C}$, stopping point ε.
Output: w

1: **while** $\sum_{x\in\mathcal{X}} |x \cdot w(t)|^2 > \varepsilon$ **do**
2: Choose pattern $x(t)$ uniformly at from $\mathcal{X}$.
3: Compute $y(t) = x(t) \cdot w(t)$.
4: Update $w(t)$ according to Eq (7).
5: $t \leftarrow t + 1$.
6: **end while**

Regularity Conditions:

A1. $\|A\|_2 \leq \Upsilon < \infty$.
A2. $\sup_{x\in\mathcal{X}} \|A_x\|_2 \leq \zeta < \infty$.
A3. $\alpha_t \geq 0, \sum \alpha_t = \infty$.
A4. $\sum \alpha_t^2 < \infty$.
A5. at each iteration t, $2\alpha_t\beta < 1$.

In the above problem, $x \in \mathcal{X}$ is a pattern drawn from the training set, β is a positive constant and $g(w)$ is the penalty term to favor sparse results. For instance one can pick $g(w) = \|w\|_1$, which leads to the ℓ_1-norm penalty, widely used in compressed sensing [14], [15]. In this paper, however, we find it more appropriate to consider $g(w) = \sum_{i=1}^{n} \tanh(\sigma w_i^2)$ for the penalty term. Intuitively, for large σ, $\tanh(\sigma w_i^2)$ approximates $|\text{sign}(w_i)|$. Therefore, the larger σ gets, the closer $g(w)$ will be to $\|\cdot\|_0$. By calculating the derivative of the objective function, and by considering the update required for each randomly picked pattern x, we will obtain the following iterative algorithm:

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

$$\tilde{w}(t+1) = w(t) - \alpha_t \left(2y(t)x(t) + \beta\Gamma(w(t))\right), \tag{4}$$

$$w(t+1) = \frac{\tilde{w}(t+1)}{\|\tilde{w}(t+1)\|_2}. \tag{5}$$

In the above equations, t is the iteration number, $x(t)$ is a pattern chosen uniformly at random from the training set $\mathcal{X}$ at iteration t , and α_t is a small positive constant. Finally, $\Gamma(w) : \mathcal{R}^n \rightarrow \mathcal{R}^n$ is the gradient $\nabla g(w)$ of the penalty term. This function has the interesting property that suppresses very small values of $w_i(t)$. To see why, consider the i-th entry of $\Gamma(w(t)))$, namely, $\Gamma_i(w(t)) = \partial g(w(t))/\partial w_i(t) = 2\sigma_t w_i(t)(1 - \tanh^2(\sigma w_i(t)^2))$. It is easy to see that $\Gamma_i(w(t)) \simeq 2\sigma w_i(t)$ for relatively small values of $w_i(t)$, and $\Gamma_i(w(t)) \simeq 0$ for larger values of $w_i(t)$. Thus, for proper choices of β and σ, Eq (4) suppresses small entries of $w(t)$ towards zero and favours sparser results.

Following the same approach as in [16] we assume that α_t to be small enough such that Eq 5 can be expanded as powers of α_t. This leads us to a simpler version of equations (3-5) as follows.

$$y(t) = x(t) \cdot w(t) \tag{6}$$

$$w(t+1) = w(t) - \alpha_t \left(y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right) + \beta\Gamma(w(t)) \right) \tag{7}$$

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

The resulting learning algorithm for one constraint node is shown in Algorithm 1. In words, $y(t)$ is the projection of $x(t)$ onto $w(t)$. If for a given data vector $x(t)$, $y(t)$ becomes zero - meaning that the data is orthogonal to the current weight vector $w(t)$ - then according to equation (7), the weight vector $w(t)$ will not be updated. However, if the data vector $x(t)$ has a non-zero projection on $w(t)$, then the weight vector will be updated in order to reduce this projection.

To prove the convergence of Algorithm 1, we benefit from the convergence of Stochastic Gradient Descent (SGD) algorithms [17]. Let $E(w) = \sum_{x\in\mathcal{X}} |x \cdot w|^2$ denote the cost function. Furthermore, let $A_x = xx^T$, and $A = \mathbb{E}\{A_x | x \in \mathcal{X}\}$ represent the correlation among patterns in the training set $\mathcal{X}$. Since patterns are uniformly sampled from $\mathcal{X}$, one can rewrite $E(w) = w^T Aw/\mathcal{C}$.

Theorem 1. *Under the regularity conditions shown above, Algorithm 1 converges to a local minimum w^* for which $\nabla E(w^*) = 0$. Furthermore, at this local minimum, the vector w^* is orthogonal to all the patterns of the training set, i.e. $Aw^* = 0$.*

The proof of this theorem is very involved (see the Appendix). In short, we rely on results of [17] to ensure convergence to a local minimum . We then show that the weight vector at this local minimum is orthogonal to the patterns of the training set. Finally, condition A5 ensures that Algorithm 1 does not converge to the trivial solution $w^* = 0$.

Algorithm 2 Intra-Module Error Correction

Input: Training set $\mathcal{X}$, threshold φ, iteration $t_{\max}$
Output: $x_1^{(\ell)}, x_2^{(\ell)}, \ldots, x_{n_\ell}^{(\ell)}$

1: **for** $t = 1 \rightarrow t_{\max}$ **do**
2: *Forward iteration:* Calculate the weighted input sum $h_i = \sum_{j=1}^{n_l} W_{ij}^{(\ell)} x_j^{(\ell)}$, for each neuron $y_i^{(\ell)}$ and set $y_i^{(\ell)} = \text{sign}(h_i^{(\ell)})$.
3: *Backward iteration:* Each neuron $x_j^{(\ell)}$ computes $g_j^{(\ell)} = \frac{\sum_{i=1}^{m_l} W_{ij}^{(\ell)} y_i^{(\ell)}}{\sum_{i=1}^{m_l} |W_{ij}^{(\ell)}|}$.
4: Update the state of each pattern neuron j according to $x_j^{(\ell)} = x_j^{(\ell)} + \text{sign}(g_j^{(\ell)})$ only if $|g_j^{(\ell)}| > \varphi$.
5: $t \leftarrow t + 1$
6: **end for**

Algorithm 3 Sequential Peeling Algorithm

Input: $\widetilde{G}, G^{(1)}, G^{(2)}, \ldots, G^{(L)}$.
Output: $x_1, x_2, \ldots, x_n$

1: **while** there is an unsatisfied $v^{(\ell)}$ **do**
2: **for** $\ell = 1 \rightarrow L$ **do**
3: If $v^{(\ell)}$ is unsatisfied, apply Algorithm 2 to cluster $G^{(l)}$.
4: If $v^{(\ell)}$ remained unsatisfied, revert the state of pattern neurons connected to $v^{(\ell)}$ to their initial state. Otherwise, keep their current states.
5: **end for**
6: **end while**
7: Declare $x_1, x_2, \ldots, x_n$ if all $v^{(\ell)}$'s are satisfied. Otherwise, declare failure.

In order to find m constraints required by the learning phase, we need to run Algorithm 1 several times. In practice, however, we can perform this process in parallel, to speed up the learning phase. It is also more meaningful from a biological point of view, as each constraint neuron can act independently from the others. Although, running the Algorithm 1 in parallel may result in redundant constraints, our experimental results show that by starting from different random initial points, the algorithm converges to distinct constraints. Besides, as long as we have enough distinct constraints, the recall algorithm in the next section works just fine and there is no need to learn all the basis vectors of the null space defined by the training set.

6 Recall Phase

The recall phase of our proposed method consists of two parts, intra-module and inter-module. In the intra-module part, each cluster tries to remove noise from its sub-pattern. As we will show in Section 6.1, this is indeed possible if the sub-patterns experience a single error. Therefor, if two errors happen in a sub-pattern, it may not be possible to correct it. To improve the error correction capability of our neural network, we then introduce the inter-module part in which clusters try to remove noise with the help of one another. What comes in our help is the overlaps between sub-patterns.

6.1 Intra-Module Recall Algorithm

In the intra-module part, we exploit the facts that the connectivity matrix of the neural network in each cluster is sparse and orthogonal to the memorized patterns. For a cluster ℓ, let $x^{(\ell)} + e^{(\ell)}$ be the given noisy input pattern. Recall that $W^{(\ell)}(x^{(\ell)} + e^{(\ell)}) = W^{(\ell)} e^{(\ell)}$.

Algorithm 2 performs a series of forward and backward iterations to remove $e^{(\ell)}$. At each iteration, the pattern neurons decide locally to whether update their current state or not: if the amount of feedback received by a pattern neuron exceeds a threshold, the neuron updates its state, and remains intact, otherwise In order to maintain the current value of a neuron, we can add self-loops to pattern neurons in Figure 1a (the self-loops are not shown in the figure for the sake of clarity).

Theorem 2. *Algorithm 2 corrects at least a single error of cluster $G^{(l)}$ with probability $1 - \left(\frac{d_{avg}^{(l)}}{m}\right)^{d_{\min}^{(l)}}$ as $\varphi \rightarrow 1$, where $d_{avg}^{(l)}$ and $d_{\min}^{(l)}$ are the average and minimum pattern-node degrees.*

The proof can be found in the Appendix. In short, we bound the probability of correcting a single error, P_{c_1}, in terms of the probability that two nodes share the same neighbourhood in a cluster.

Theorem 2 implies that $\Lambda^{(l)}(\frac{d_{\text{avg}}}{m})$ should be negligible in order for Algorithm 2 to correct at least one error. For settings considered in this paper, we have $d_{\min}^{(l)} \geq 3$ and $d_{\text{avg}}^{(l)} \leq 0.1m$, which yields to $P_{c_1} \geq 0.999$. To simplify the analysis of Section 6.2, we make a conservative assumption that a

cluster is capable of correcting only a single error with overwhelming probability, without introducing additional errors. In practice, as it is confirmed by our simulations, clusters are to able to correct more than one error.

In practice, we replace the conditions $h_i = 0$ and $|h_i| > 0$ with $|h_i| < \varepsilon$ and $|h_i| > \varepsilon$, respectively, for some small positive number ε. To obtain the best possible result, one should tune the update parameter φ. In this paper, we opt for the conservative choice $\varphi = 1$, which in turn increases the error correction time, but ensures higher error correction probability and simplifies the analysis.

As stated earlier, the efficiency of Algorithm 2 relies on the assumption that the neural network is sparse. To gain some insight, consider an extreme case where the bipartite graph is complete. Then, a single error results in the violation of all constraint neurons in the forward iteration. Therefore, in the backward iteration, all the pattern neurons receive feedback from their neighbours. This make it impossible to tell which pattern neuron is the noisy one. On the other hand, once the graph is sparse, a single error makes only a small set of constraint neurons unsatisfied. Consequently, in the backward iteration, only the pattern nodes that share the same neighbourhood with the noisy one receive feedback. Note that in this case, the fraction of the received feedback will be much larger for the true noisy neuron. Therefore, by merely looking at the fraction of received feedback from the constraint neurons, one can identify the noisy pattern neuron.

6.2 Inter-Module Recall Algorithm

In the previous section, we showed that a single error inside a cluster can be corrected with high probability. In fact, a finer analysis reveals that there exists a threshold $T_e > 1$ such that any number of errors less than T_e can be corrected. Since clusters have overlaps with one another, removing an error from a cluster, can potentially help the others.

Our proposed algorithm is based on a famous error correction decoder called *peeling algorithm*. To begin, let us reconfigure the bipartite graph G as follows. For each cluster $G^{(l)}$, we contract its set of constraint nodes $y_1^{(l)}, y_2^{(l)}, \ldots, y_{m_l}^{(l)}$ into a single node $v^{(l)}$. This will lead to a new bipartite graph $\widetilde{G}$ whose degree distribution obviously changes (see Figure 1b). Let us denote the degree distributions of $\widetilde{G}$ (from the edge perspective) by $\widetilde{\lambda}$ and $\widetilde{\rho}$. We say that the node $v^{(l)}$ is unsatisfied if it is connected to a noisy pattern node. The asymptotic performance of our error recovery method, shown in Algorithm 3 is given by the following theorem.

Theorem 3. *Under the assumptions that graph $\widetilde{G}$ grows large and it is chosen randomly with degree distributions given by $\widetilde{\lambda}$ and $\widetilde{\rho}$, Algorithm 3 is successful if $p_e \cdot \widetilde{\lambda}(1 - \widetilde{\rho}(1 - z)) < z$ for $z \in (0, p_e)$.*

The proof is based on the density evolution technique [18, 19], detailed in the Appendix.

The condition given in Theorem 3 can be used to calculated the maximal fraction of errors Algorithm 3 for the given degree distributions can correct. For instance, for the the degree distribution pair $(\widetilde{\lambda}(z) = z^2, \widetilde{\rho}(z) = z^5)$, the threshold is $p_e \approx 0.429$, below which Algorithm 3 corrects all the errors with high probability. Note that the predicted threshold by Theorem 3 is based on the assumption that a cluster can only correct a single error. In practice, as we noted earlier, a cluster can correct more. Hence, the threshold predicted by Theorem 3 is a lower bound on the overall recall performance of our neural network (as the size of the network grows).

7 Pattern Retrieval Capacity

Note that the number of patterns $\mathcal{C}$ does not have any effect on the learning or recall algorithm except for its obvious influence on the learning time. A long as the patterns come from a subspace, the learning algorithm (Algorithm 1) will yield a matrix W which is orthogonal to all the patterns of the training set. In the recall phase, the proposed denoising algorithms (Algorithms 2 and 3) deal only with $W \cdot e$, where the entries of e are sampled independently from the patterns. In order to show that the pattern retrieval capacity is exponential in n, all we need to demonstrate is that there exists a training set $\mathcal{X}$ with $\mathcal{C}$ patterns of length n for which $\mathcal{C} \propto a^{rn}$, for some $a > 1$ and $0 < r < 1$.

Theorem 4. *With slight abuse of notation, let $\mathcal{X}$ be a $\mathcal{C} \times n$ matrix, formed by C vectors of length n with entries from the set $\mathcal{S}$. Furthermore, let $k = rn$ for some $0 < r < 1$. Then, there exists a set of vectors for which $\mathcal{C} = a^{rn}$, with $a > 1$, and rank$(\mathcal{X}) = k < n$.*

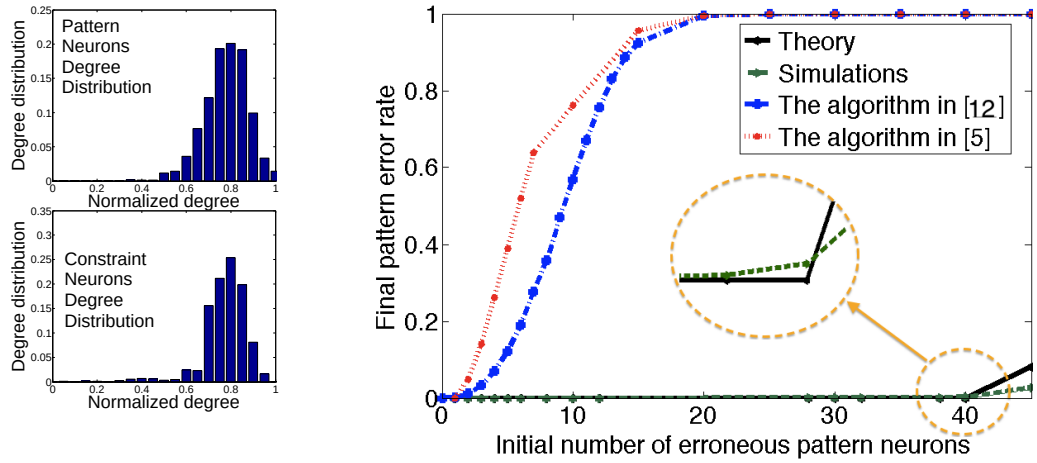


Figure 2: Pattern and constraint neuron degree distributions for $n = 400$, $L = 50$, and on average of 20 constraints per cluster. The learning parameters are $\alpha = 0.95\beta = 0.75$, and $\theta = 0.05$.

The proof of this theorem is by construction. The details are outlined in the Appendix.

8 Simulation Results

To simulate, there is a systematic way of generating patterns satisfying a set of linear constraints. The details can be found in the constructive proof of Theorem 4 as well as in [5, 12]. In our simulations, we assume that each pattern neuron is connected to around 5 clusters.

In the learning phase, Algorithm 1 is run in parallel which results in learning the constraints of each cluster. In the recall phase, at each round, a pattern x is sampled uniformly at random from the training set. Then, each of its entries gets corrupted independently with probability p_e. Afterwards, Algorithm 3 tries to denoise the corrupted pattern. When finished, we declare an error if $\hat{x} \neq x$, where $\hat{x}$ is the Algorithm 3's output. We repeat this process several times to calculate the error rate, and compare it to the bound derived in section 6.2.

Learning Results: The left panels in Figures 2 illustrate the degree distributions of pattern and constraint neurons, respectively, over an ensemble of 5 randomly generated simulation setups. The horizontal axis shows the normalized degree of pattern (resp., constraint) neurons and the vertical axis represents the fraction of neurons with the given normalized degree. The parameters for the learning algorithm are $\alpha = 0.95$, $\beta = 0.75$, and $\theta = 0.05$.

We have executed the learning algorithm for different network sizes and learning parameters. Due to space limitation the results are not reported here (qualitatively, they all look similar to the above figure). Interestingly, in almost all cases, the learning phase converged within two learning iterations, i.e. by going over the data set only twice.

Recall Results: The right panel of Figure 2 illustrates the performance of the recall algorithm in the presence of noise. The horizontal axis represents the number of initial erroneous neurons. The vertical axis shows the final pattern error rate. The performance is compared with the theoretical bound derived in section 6.2, as well as the results of the algorithms proposed in [12] and [5]. The parameters used for this simulation are $n = 400$, $L = 50$ and $\varphi = 0.82$ in the recall algorithm 2. For [12], the network size is $n = 400$ with 4 clusters in the first level and one cluster in the second level. Note the slight difference between the theoretical and simulation results in the low noise regime (emphasized in the right panel of Figure 2), which is due to the following facts: 1) we stop Algorithm 3 after a limited number of iterations, $t_{\max}$, and 2) the network size is small. The threshold predicted by Theorem 3 becomes accurate as $n \to \infty$.

References

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

[2] 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.

[3] 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.

[4] 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.

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

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

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

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

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

[10] 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.

[11] 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.

[12] A. H. Salavati, A. Karbasi, *Multi-Level Error-Resilient Neural Networks*, To appear in the IEEE Int. Sym. Inf. Theory (ISIT 2012), 2012.

[13] E. Candes, 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.

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

[15] 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.

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

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

[18] Luby, M.G. and Mitzenmacher, M. and Shokrollahi, M.A. and Spielman, D.A., *Efficient erasure correcting codes*, IEEE Trans on Information Theory, Vol 47, 2001.

[19] T. Richardson, R. Urbanke *Modern Coding Theory*, Cambridge University Press, 2008.

Proof of Theorem 1

To prove the lemma, we use the convergence results in [17] and show that the required assumptions to ensure convergence holds for the proposed algorithm. For simplicity, these assumptions are listed here:

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

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

Assumption 3 ensures that the vector w could not escape by becoming larger and larger. Due to the constraint $\|w\|_2 = 1$, this assumption holds as well.

Assumption 4 holds as well because:

$$\begin{aligned}
\mathbb{E}_x \left(2A_x w + \lambda\Gamma(w)\right)^2 &= 4w^T \mathbb{E}_x(A_x^2) w + \lambda^2 \|\Gamma(w)\|_2^2 \\
&+ 4\lambda w^T \mathbb{E}_x(A_x)\Gamma(w) \\
&\leq 4\|w\|_2^2 \zeta^2 + \lambda^2 \|w\|_2^2 + 4\lambda\Upsilon\|w\|_2^2 \\
&= \|w\|_2^2 (4\zeta^2 + 4\lambda\Upsilon + \lambda^2) \qquad (8)
\end{aligned}$$

Finally, assumption 5 holds because:

$$\begin{aligned}
\|2A_x w + \lambda\Gamma(w)\|_2^2 &= 4w^T A_x^2 w + \lambda^2 \|\Gamma(w)\|_2^2 \\
&+ 4\lambda w^T A_x \Gamma(w) \\
&\leq \|w\|_2^2 (4\zeta^2 + 4\lambda\zeta + \lambda^2) \qquad (9)
\end{aligned}$$

Therefore, $\exists F > D$ such that as long as $\|w\|_2^2 < F$:

$$\sup_{\|w\|_2^2 < E} \|2A_x w + \lambda\Gamma(w)\|_2^2 \leq (2\zeta + \lambda)^2 F = \text{constant} \qquad (10)$$

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

Next, we prove the desired result, i.e. the fact that in the local minimum, the resulting weight vector is orthogonal to the patterns, i.e. $Aw = 0$. Since $\nabla E(w^*) = 2Aw^* + \lambda\Gamma(w^*) = 0$, we have:

$$w^* \cdot \nabla E(w^*) = 2(w^*)^T A w^* + \lambda w^* \cdot \Gamma(w^*) \qquad (11)$$

The first term is always greater than or equal to zero. Now as for the second term, we have that $|\Gamma(w_i)| \leq |w_i|$ and $\text{sign}(w_i) = \text{sign}(\Gamma(w_i))$, where w_i is the i^{th} entry of w. Therefore, $0 \leq$

$w^* \cdot \Gamma(w^*) \leq \|w^*\|_2^2$. Therefore, both terms on the right hand side of (11) are greater than or equal to zero. And since the left hand side is known to be equal to zero, we conclude that $(w^*)^T A w^* = 0$ and $\Gamma(w^*) = 0$. The former means $(w^*)^T A w^* = \sum_{x \in \mathcal{X}} (w^* \cdot x)^2 = 0$. Therefore, we must have $w^* \cdot x = 0$, for all $x \in \mathcal{X}$. Which simply means the vector w^* is orthogonal to all the patterns in the training set.

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

To this end, denote $w'(t) = w(t) - \alpha_t y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right)$ and consider the following inequalities:

$$\begin{aligned} \|w(t+1)\|_2^2 &= \|w(t) - \alpha_t y(t) \left(x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2} \right) - \alpha_t \lambda \Gamma(w(t))\|_2^2 \\ &= \|w'(t)\|^2 + \alpha_t^2 \lambda^2 \|\Gamma(w(t))\|^2 - 2\alpha_t \lambda \Gamma(w(t)) \cdot w'(t) \\ &\geq \|w'(t)\|_2^2 - 2\alpha_t \lambda \Gamma(w(t)) \cdot w'(t) \end{aligned} \tag{12}$$

Now in order to have $\|w(t+1)\|_2^2 > 0$, we must have that $2\alpha_t \lambda |\Gamma(w(t))^T w'(t)| \leq \|w'(t)\|_2^2$. Given that, $|\Gamma(w(t)) \cdot w'(t)| \leq \|w'(t)\|_2 \|\Gamma(w(t))\|_2$, it is therefore sufficient to have $2\alpha_t \lambda \|\Gamma(w(t))\|_2 \leq \|w'(t)\|_2$. On the other hand, we have:

$$\begin{aligned} \|w'(t)\|_2^2 &= \|w(t)\|_2^2 + \alpha_t^2 y(t)^2 \|x(t) - \frac{y(t)w(t)}{\|w(t)\|_2^2}\|_2^2 \\ &\geq \|w(t)\|_2^2 \end{aligned} \tag{13}$$

As a result, in order to have $\|w(t+1)\|_2^2 > 0$, it is sufficient to have $2\alpha_t \lambda \|\Gamma(w(t))\|_2 \leq \|w(t)\|_2$. Finally, since we have $|\Gamma(w(t))| \leq |w(t)|$ (entry-wise), we know that $\|\Gamma(w(t))\|_2 \leq \|w(t)\|_2$. Therefore, having $2\alpha_t \lambda < 1 \leq \|w(t)\|_2 / \|\Gamma(w(t))\|_2$ ensures $\|w(t)\|_2 > 0$.

Proof of Theorem 2

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 of cluster ℓ is the noisy one. Furthermore, let $z = \{1, \ldots, 0\}$ be the noise vector. Denoting the i^{th} column of the weight matrix by $W_i^{(\ell)}$, we will have $y^{(\ell)} = \text{sign}(W_1^{(\ell)})$. 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 $W_i \cdot \text{sign}(z_1 W_1^{(\ell)}) = \|W_i^{(\ell)}\|_0$. Note that $W_i^{(\ell)} \cdot \text{sign}(W_1^{(\ell)}) < \|W_i^{(\ell)}\|_0$ 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 $\mathcal{N}(x_i) \cap \mathcal{N}(x_1) \neq \mathcal{N}(x_i)$, there are non-zero entries in $W_i^{(\ell)}$ while $W_1^{(\ell)}$ is zero and vice-versa. Therefore, letting P'_{x_i} being the probability of $\mathcal{N}(x_i) \cap \mathcal{N}(x_1) \neq \mathcal{N}(x_i)$, 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, $d_{\text{avg}}^{(l)} = \sum_i \Lambda_i d_i$ be the average degree of pattern neurons and finally $d_{\text{avg}}^{(l)}$ be the minimum degree of pattern neurons. Then, we know that a noisy pattern neuron is connected to $d_{\text{avg}}^{(l)}$ constraint neurons on average. Therefore, the probability of x_i and x_1 share exactly the same neighborhood would be:

$$P'_{x_i} = \left(\frac{d_{\text{avg}}^{(l)}}{m} \right)^{d_{x_i}} \tag{14}$$

Taking the average over the pattern neurons, we have

$$\begin{aligned} P'_e &= \Pr\{x \in C_t\}\mathbb{E}_{d_{x_i}}\{P'_{x_i}\} \\ &= (1-\frac{1}{n})\Lambda(\frac{d_{\text{avg}}^{(l)}}{m}) \\ &= \Lambda(\frac{d_{\text{avg}}^{(l)}}{m}) \end{aligned} \tag{15}$$

where 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

$$\begin{aligned} P_c &\geq 1 - \Lambda(\frac{d_{\text{avg}}^{(l)}}{m}) \\ &\geq 1 - \left(\frac{d_{\text{avg}}^{(l)}}{m}\right)^{d_{\text{avg}}^{(l)}} \end{aligned} \tag{16}$$

Proof of Theorem 3

The proof is similar to Theorem 3.50 in [19]. Each cluster node receives an error message from its neighboring pattern nodes with probability z. Now consider a given *noisy* pattern neuron which is connected to a given cluster $v^{(\ell)}$. Let $\pi^{(\ell)}(t)$ be the probability that the cluster node $v^{(\ell)}$ with degree $\widetilde{d}_\ell$ sends an error message during iteration t of Algorithm 3. This event happens if the cluster node $v^{(\ell)}$ receives at least one error message from its other neighbors among pattern neurons along its input edges, i.e. if it is connected to more than one noisy pattern neuron. Therefore,

$$\pi^{(\ell)}(t) = 1 - (1 - z(t))^{\widetilde{d}_\ell - 1} \tag{17}$$

As a result, if $\pi(t)$ shows the average probability that a cluster node sends a message declaring the violation of at least one of its constraint neurons, we will have,

$$\pi(t) = \mathbb{E}_{\widetilde{d}_\ell}\{\pi^{(\ell)}(t)\} = \sum_i \widetilde{\rho}_i (1 - (1 - z(t))^{\widetilde{d}_\ell - 1}) = 1 - \widetilde{\rho}(1 - z(t)) \tag{18}$$

Now consider a given pattern neuron x_i with degree d_i. This node will remain noisy in iteration $t+1$ of Algorithm 3 if it was noisy in the first place and in iteration $t+1$ all of its neighbors among constraint neurons send a violation message. Therefore, the probability of this node being noisy will be $z(0)\pi(t)^{d_i}$. As a result, noting that $z(0) = p_e$, the average probability that a pattern neurons remains noisy will be

$$z(t+1) = p_e \cdot \sum_i \widetilde{\lambda}_i \pi(t)^i = p_e \cdot \widetilde{\lambda}(\pi(t)) = p_e \cdot \widetilde{\lambda}(1 - \widetilde{\rho}(1 - z(t))) \tag{19}$$

Therefore, the decoding operation will be successful if $z(t+1) < z(t),\ \forall t$. As a result, we must look for the maximum p_e such that we will have $p_e \cdot \widetilde{\lambda}(1 - \widetilde{\rho}(1 - z)) < z$ for $z \in [0, p_e]$.

Proof of Theorem 4

The proof is based on construction: we construct a data set $\mathcal{X}$ with the required properties such that it can be memorized by the proposed neural network.

To start, consider a matrix $G \in \mathbb{R}^{k \times n}$ with rank k and $k = rn$, with $0 < r < 1$. Let the entries of G be non-negative integers, between 0 and $\gamma - 1$, with $\gamma \geq 2$.

We start constructing the patterns in the data set as follows: consider a random vector $u \in \mathbb{R}^k$ with integer-valued-entries between 0 and $\upsilon - 1$, where $\upsilon \geq 2$. We set the pattern $x \in \mathcal{X}$ to be $x = u \cdot G$, *if* all the entries of x are between 0 and $S-1$. Obviously, since both u and G have only non-negative

entries, all entries in x are non-negative. Therefore, it is the $S-1$ upper bound that we have to worry about.

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

$$x_j = u \cdot g_j \leq d_j(\gamma - 1)(\upsilon - 1)$$

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

$$S - 1 \geq d^*(\gamma - 1)(\upsilon - 1) \tag{20}$$

to ensure all entries of x are less than S.

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

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

Remark 1. *Note that the inequality (20) was obtained for the worst-case scenario and in fact is very loose. Therefore, even if it does not hold, we will still be able to memorize a very large number of patterns since a big portion of the generated vectors x will have entries less than S. These vectors correspond to the message vectors u that are "sparse" as well, i.e. do not have all entries greater than zero. The number of such vectors is a polynomial in n, the degree of which depends on the number of non-zero entries in u.*