Nearly-optimal associative memories based on distributed constant weight codes

Vincent Gripon
Electronics and Computer Enginering
McGill University
Montréal, Canada
Email: vincent.gripon@ens-cachan.org

Claude Berrou
Electronics Department
Télécom Bretagne
Brest, France
Email: claude.berrou@telecom-bretagne.eu

***Abstract*—A new family of sparse neural networks achieving nearly optimal performance has been recently introduced. In these networks, messages are stored as cliques in clustered graphs. In this paper, we interpret these networks using the formalism of error correcting codes. To achieve this, we introduce two original codes, the thrifty code and the clique code, that are both sub-families of binary constant weight codes. We also provide the networks with an enhanced retrieving rule that enables a property of answer correctness and that improves performance.**

***Index Terms*—associative memory, classification, constant weight codes, clique code, thrifty code, sparse neural networks, winner-take-all**

I. Introduction

One can split the family of memories into two main branches. The first one contains indexed memories. In an indexed memory, data messages are stored at specific indexed locations. Thus, messages are not overlapping, and directly accessing a stored message requires to know its address. It is a convenient paradigm as far as data itself is not useful *a priori*. For example, a postman just needs to know your address to bring you mails, and does not care about the content of the mail nor the color of your front door.

The second branch is that of associative memories. An associative memory is such that a previously learned message can be retrieved from part of its content. It is tricky to define how large is the "part" of the content that is necessary to retrieve the data. A reasonable definition is to consider this "part" to be close to the minimum amount of data required to unambiguously address a unique previously learned message. Contrary to indexed memories, it is likely that messages overlap one another in associative memories. This paradigm is convenient when trying to find data from other data. For example, a detective might be interested in remembering the name of that woman he questioned who owns a car of the same brand as that of the murderer.

It is obviously possible to simulate one memory using the other if given unlimited computational power. Indeed, to obtain an associative memory, one can read all the stored messages in an indexed memory and compare them with the part of messages it is given as input. It then selects the one that matches the best the input. Conversely, one can learn simultaneously a message and its associated index into an associative memory so that messages can be retrieved from indexes. Yet emulating one memory using the other may prove to be inefficient in practice.

Actually, associative memories previously were not scalable: algorithm based techniques are too computationally greedy, while specifically designed architectures provided a poor efficiency - the ratio of the amount of memory learned to the amount of memory required by the implementation of the architecture. As a matter of fact, the state-of-the-art architecture, in terms of efficiency, used to be the Hopfield neural network [1] which efficiency tends to zero as the amount of learned messages tends to infinity [2]. Recently Gripon and Berrou have introduced a new model of sparse neural networks (called GBNNs in [3]) that addresses the previous limitations in terms of efficiency and provides nearly-optimal associative memories [4], [5]. The way these memories function is inspired by that of error correcting decoders [6]. Indeed, in the case of an erasure transmission channel, both systems aim at retrieving data from part of it. Yet the analogy is not that simple as a decoder does not have the ability to learn new codewords whereas memories are precisely designed to learn messages.

The architecture of these memories mimics that of distributed codes, like turbo codes [7] and LDPC codes [8], [9]. As a matter of fact, the codes they use locally are weak but are associated together to provide a globally efficient one.

In this document, we interpret these new associative memories under the formalism of error correcting codes. Then, we propose a new retrieving rule that enhances performance and ensures answers correctness.

In section II we introduce two original sub-families of binary constant weight codes: the thrifty code and the clique code. In section III, we present the GBNN functioning. Section IV is dedicated to introducing the new retrieving rule and finally we discuss performance in section V.

II. Binary constant weight codes

Binary constant weight codes [10] are a family of error correcting codes such that all their codewords have the same weight, the weight of a binary ($\{0;1\}$) word being the number of 1s it contains.

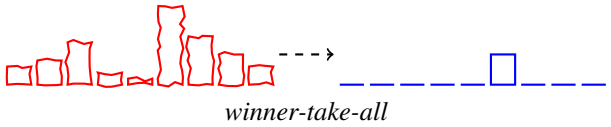

Figure 1. Illustration of the decoding of thrifty code using the winner-take-all rule.

Such a code is determined by a tuple $< n, r, w >$ where n is the length of the codewords, r is the overlapping value and w the weight of the codewords. The overlapping value r is the maximum number of 1s two distinct codewords can have in common. It is directly connected to the minimum Hamming distance d_m (the minimum number of distinct symbols between two distinct codewords) of the code. This relation is given by the formula:

$$d_m \geq 2(w - r)$$

For example, a degenerated binary constant weight code with $r = 0$ has minimum Hamming distance at least $2w$, which is exactly the Hamming distance between any distinct codewords in this code. The minimum Hamming distance of a code is a very important parameter to assess performance, as the maximum number of erasures e_{max} an error correcting code can retrieve is:

$$e_{max} = d_m$$

A. The thrifty code

We call thrifty codes the sub-family of binary constant weight codes with weight $w = 1$. The term "thrifty" comes from the fact the codewords contain a single 1 each. Due to this restricted weight, the overlapping parameter r has no influence on the code.

Let us define a maximum thrifty code as a thrifty code that cannot be completed by any codeword without breaking the constant weight rule. In other words, the unique maximum thrifty code of length n is $\{0^k 10^{n-1-k}, 0 \leq k \leq n-1\}$ whereas all its subsets are thrifty codes. The maximum thrifty code is not a powerful one. Indeed, its minimum Hamming distance is 2. On the other hand, it can be very easily decoded.

As an example, let us consider that a codeword has been transmitted through a transmission channel that affects every symbol according to the function:

$$f : \begin{cases} \{0; 1\} \to \mathbb{R} \\ x \mapsto x + \theta \end{cases}$$

where θ is an independent random variable. Then the most likely transmitted codeword is the one with a 1 where the received value is maximum. Figure 1 illustrates this decoding rule that corresponds to what biologists call the winner-take-all rule.

Moreover, the fact a code has a small minimum Hamming distance does not make it useless. LDPC codes, for example, are based on parity codes, which minimum Hamming distance is also 2.

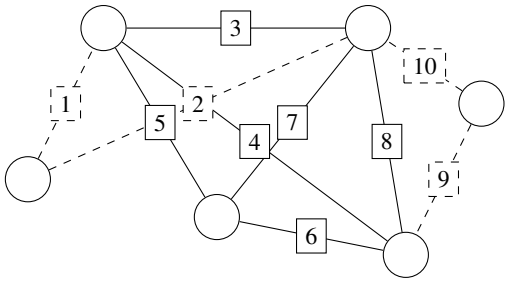

Figure 2. Example of a clique in a graph with no loop. Circles represent vertices and squares contain the labels of connections. The clique is represented by the straight lines.

In order to aggregate thrifty codes, we will now introduce a more powerful one: the clique code.

B. The clique code

A code is basically a constraint on words. Based on this idea, one can associate cliques of constant size in a graph with codewords.

Let us first recall that a clique in a graph with no loop is a set of vertices that are fully interconnected. Such a clique can be represented by the set of its connections. Consider Figure 2 for example, where a clique is represented with straight lines. This clique corresponds to the set of connections $\{3; 4; 5; 6; 7; 8\}$. Conversely, some subsets of connections do not correspond to a clique in the graph. For example the set $\{1; 2; 3; 4; 5; 6\}$ does not represent a clique. This emphasizes the fact cliques form a constraint on the subsets of connections, that is to say a code.

To make a connection with the binary constant weight codes, such a clique can be represented by a binary vector containing 1s at the coordinates of the connections of the clique. The previous example would thus give the binary codeword 0011111100. Based on this representation, to each clique containing c vertices in the graph corresponds a binary vector containing $\binom{c}{2}$ 1s. Moreover, the Hamming distance between the binary representations of two distinct cliques of the same size c is at least $2\binom{c-1}{2}$, corresponding to the case where the two cliques have $c - 1$ vertices in common. Hence, contrary to the thrifty code, non trivial clique codes can have an arbitrarily large minimum Hamming distance. In conclusion, the binary representation of cliques of constant size c in a graph with no loop and with n vertices are codewords of a binary constant weight code with parameters $< \binom{n}{2}, \binom{c}{2}, \binom{c}{2} - \binom{c-1}{2}, \binom{c}{2} >$.

III. GBNNs

In [4], [5], Gripon and Berrou introduced a new architecture for neural networks that addresses the previous limitations of associative memories in terms of efficiency (GBNNs). The following subsections introduce the learning and retrieving rules.

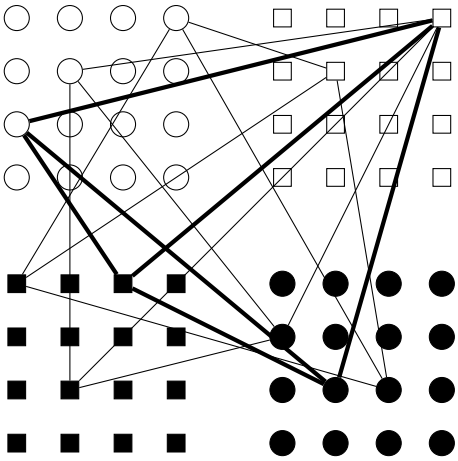

Figure 3. Representation of the learning of a new message into the network. The network is split into 4 clusters (empty circles, empty squares, filled circles and filled squares) made of 16 neurons each. The thick connections represent the clique associated with the message to learn that is being added to the network.

A. Learning

GBNNs main idea is to aggregate thrifty codes together using a powerful clique code. To describe this neural network, let us introduce some parameters first: the network contains n neurons divided into c clusters made of $l = \frac{n}{c}$ neurons each.

Such a network can learn l-ary messages of length c. A message m to learn is thus made of c symbols $m_1, m_2, \ldots, m_c$, associated one to one with the clusters of the network. Let us fix such a message.

At the scale of cluster, the symbol the message corresponds to is represented by a unique neuron. This representation corresponds to that of the thrifty code. At the scale of the network, it means that a message m is represented by a single neuron in each cluster. To learn such a message, every possible connection between those selected neurons are added to the network, printing a clique into it - with the exception of loops. An already existing connection is not reinforced, which means that the network is binary (either a connection exists or not). Figure 3 depicts the learning of a message. In this representation, the thick connections are forming a clique that corresponds to the message to learn and are being printed into the network.

The amount of messages the network can learn before being overloaded is directly connected to its density, that is the ratio of the number of created connections to that of possible ones. In case of learning messages independently and identically distributed, the density can be fairly estimated using the formula:

$$d = 1 - (1 - \frac{1}{l^2})^M$$

where M is the number of learned messages. Let us insist that in this case the density d does not depend on the number of clusters c. As a direct consequence, the network will give better results having a few large clusters rather than a lot of small ones.

After the learning of all the messages, the network can be used to retrieve any of them from part of its content.

B. Retrieving

Let us consider that a previously learned message m is presented as input to such a network, but with missing symbols. After selecting the correct neurons in the clusters where the symbols of the message are not missing (no one elsewhere), the retrieving process begins. This process iterates two phases.

The first phase corresponds to the decoding of the global clique code. During this phase, the selected neurons send signals through the connections they are connected to. Thus, conversely, each neuron of the network receives a number of incoming signals, possibly zero.

The second phase corresponds to the local thrifty code. During this phase, in each cluster is selected the neuron or the group of neurons that achieves the maximum number of incoming signals. In other terms, in each cluster is applied the winner-take-all rule. To obtain good performance, a memory effect denoted γ is added to the number of incoming signals before the winner-take-all rule, increasing the scores of previously selected neurons. In associative memory applications, the success of retrieving rate of the network increases with the memory effect value.

The network converges in a few iterations. At the end of the process, the clusters which originally did not have any selected neuron are provided with the selection of a neuron or a group of neurons. Those neurons are the answer of the network.

Let us introduce $v^t_{(c_i,l_j)}$ such that $v^t_{(c_i,l_j)} = 1$ if the l_jth neuron of the c_ith cluster is selected at iteration t and $v^t_{(c_i,l_j)} = 0$ otherwise. Let us extend v^t such that iteration 0 corresponds to the neurons selected from the non-missing symbols in the input. Let us introduce the adjacency matrix coefficient of the network $W_{(c_i,l_j),(c_{i'},l_{j'})}$ which equals 1 if the connection between the l_jth neuron of the c_ith cluster is connected to the $l_{j'}$th neuron of the $c_{i'}$th cluster and 0 otherwise. Then the retrieving process can be formalized as:

$$\begin{aligned}
s^t_{(c_i,l_j)} &= \gamma v^t_{(c_i,l_j)} + \\
&\quad \sum_{c_{i'}=1}^{c} \sum_{l_{j'}=1}^{l} v^t_{(c_{i'},l_{j'})} \cdot W_{(c_{i'},l_{j'})(c_i,l_j)} \qquad (1)\\
s^t_{max,c_i} &= \max_{1 \le l_j \le l} [s^t_{(c_i,l_j)}] \\
v^{t+1}_{(c_i,l_j)} &= \begin{cases} 1 & \text{if } s^t_{(c_i,l_j)} = s^t_{max,c_i} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

IV. Improving the retrieving process

A. Improving performance

The previously described retrieving process suffers from a major drawback. Indeed, a cluster with an ambiguous selection - that is several neurons selected - has potentially a more important impact on the decoding in the other clusters than that of an unambiguous cluster.

For example, let us consider that cluster c_1 has two of its selected neurons connected to neuron l_1 in cluster c_2. Thus l_1 will receive at least two signals from cluster c_1, reinforcing the probability to be selected. Cluster c_2 contains another neuron l_2 which is receiving all its signals from distinct clusters and that achieves the same score as l_1. Thus, if l_2 is selected then so will be l_1. However, l_2 receives signals from connections that may belong to the same clique, that is to say representing a previously learned message. On the other hand, l_1 is adding signals from connections that cannot belong to the same clique. Therefore, l_2 should be favored with respect to l_1.

Let us modify the retrieving process to take this remark into account: first, let us modify Equation (1) such that:

$$s^t_{(c_i,l_j)} = \gamma v^t_{(c_i,l_j)} + \sum_{c_{i'}=1}^{c} \max_{1\leq l_{j'}\leq l} v^t_{(c_{i'},l_{j'})} \cdot W_{(c_{i'},l_{j'})(c_i,l_j)}$$

which means that the score of a neuron will not be larger if it receives two or more signals from the same cluster.

With this new equation, an ambiguous cluster will no longer have more impact on the decision of others. To take full advantage of this new technique, one can also decide to select all neurons in clusters corresponding to missing symbols (instead of no one). Consequently, the score of the correct neurons in each cluster is now known *a priori*. Indeed, as they will receive connections from all the other neurons that correspond to the message to retrieve, they will achieve exactly score $\gamma + c - 1$.

Taking these two modifications into account, the overall retrieving process can be synthesized as follows:

$$v^{t+1}_{(c_i,l_j)} = \begin{cases} 1 & \text{if } \gamma v^t_{(c_i,l_j)} + \\ & \sum_{c_{i'}=1}^{c} \max_{1\leq l_{j'}\leq l} v^t_{(c_{i'},l_{j'})} W_{(c_{i'},l_{j'})(c_i,l_j)} = \gamma + c - 1 \\ 0 & \text{otherwise} \end{cases}$$

B. Answers correctness

GBNNs have two ways to fail in retrieving a message. The first one is giving a wrong answer, the second one is giving an ambiguous answer. Thanks to the new retrieving rule, the network can no longer give a wrong answer. In other words, an unambiguous answer will always be correct. Let us first show how the old retrieving rule can result in a wrong decision.

Let us consider the following example: the network is split into 3 clusters made of 3 neurons each. The messages it learns are therefore made of three ternary symbols, $\{1; 2; 3\}$ for instance. Let us consider that the network has learned the three following messages and no other: 111, 112 and 123. Figure 4 depicts the connections of this network.

If the partial message 1** is presented to the network, where * denotes a missing symbol, then the network should not take any decision. As a matter of fact, the problem is intrinsically ambiguous. Yet, the retrieving process described in [4], [5] will lead to a wrong answer.

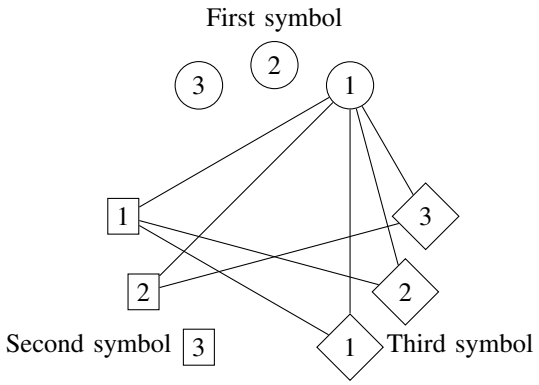

Figure 4. Representation of the connections in the network after the learning of messages 111, 112 and 123.

To understand this phenomenon, let us follow this process step by step.

At the end of the first iteration, neuron 1 in cluster 1, neurons 1 and 2 in cluster 2 and all three neurons of cluster 3 will be selected for the next iteration, considering $\gamma \neq 0$.

At iteration 2, neuron 1 in cluster 2 will receive 3 incoming signals (respectively from neuron 1 in cluster 1 and neurons 1 and 2 in cluster 3) whereas neuron 2 in the same cluster will only receive 2 incoming signals. Therefore, neuron 1 will be the only one selected in cluster 2 at the end of iteration 2.

Actually, this selection will not evolve during the next iterations, such that the network will give 1 as the answers for the second symbol of the message to retrieve. This choice is arbitrary and may lead to a wrong global answer.

This cannot happen with the rule we introduce in this paper. Indeed, all the possible learned messages - according to the given partial input - will be such that their corresponding neurons will achieve the maximum score. Therefore, the winner-take-all rule will select all of them and the network will not give a wrong answer.

V. Performance

Figure 5 depicts the retrieval error rate of networks made of 8 clusters of 256 neurons each when 4 symbols are missing in inputs and as a function of the number of learned messages. This figure emphasizes the gain in performance obtained with the new retrieving rule.

Figure 6 depicts the ratio of wrong answers given by networks with the same architectures as in Figure 5 and with the old retrieving rule (this ratio is always 0 with the new rule).

VI. Conclusion

In this paper we introduced GBNNs using error correcting codes formalism. By doing this, we showed how the principles of distributive codes can be enlarged to other domains, such as associative memories. This extension required the introduction of two original codes: the thrifty code and the clique code, which are two sub-families of binary constant weight codes.

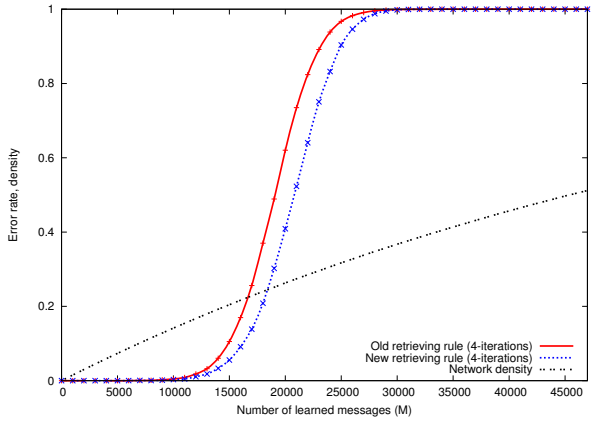

Figure 5. Evolution of the message retrieval error rate in a network made of 8 clusters of 256 neurons each when input messages are missing 4 symbols and as a function of the number of learned messages.

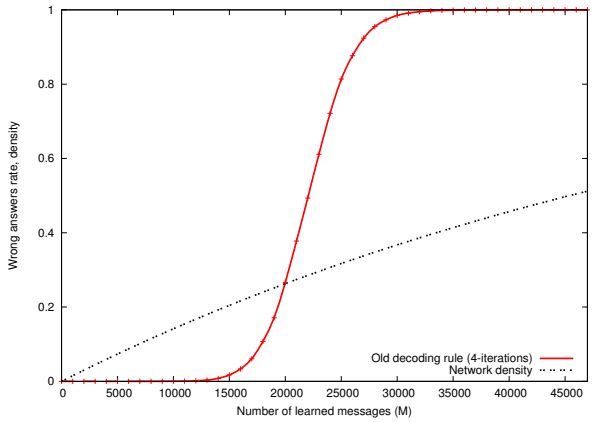

Figure 6. Evolution of the wrong answers rate in a network made of 8 clusters of 256 neurons each when input messages are missing 4 symbols and as a function of the number of learned messages.

We introduced a new retrieving rule that significantly improves performance with regards to the previous one. This rule also provides the network with an answer correctness property. Along with this new rule, GBNNs make a supplementary step towards optimality.

One possible immediate application of this new rule would be to reduce the area consumption in the implementation of GBNNs [3].

Future work will also include addressing the limitation on the network density that does not depend on the number of clusters, making it unsuitable for the learning of long messages.

REFERENCES

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational properties," *Proc. Natl Acad. Sci., Biophysics*, vol. 79, pp. 2554–2558, USA, 1982.

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

[3] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, "Architecture and implementation of an associative memory using sparse clustered networks," to appear in *IEEE International Symposium on Circuits and Systems*, 2012.

[4] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *IEEE transactions on neural networks*, vol. 22, pp. 1087 – 1096, Jul. 2011.

[5] V. Gripon and C. Berrou, "A simple and efficient way to store many messages using neural cliques," *Proc. of IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain*, pp. 54 – 58, Apr. 2011.

[6] V. Gripon, *Networks of neural cliques*. PhD thesis, Télécom Bretagne, July 2011.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," *Proc. of IEEE ICC '93*, vol. 2, pp. 1064–1070, Geneva, May 1993.

[8] R. Gallager, "Low-density parity-check codes," *IEEE Trans. on Inf. Theory*, vol. 8, pp. 21–28, Jan. 1962.

[9] D. J. MacKay and R. M. Neal, "Good codes based on very sparse matrices," *Cryptography and Coding. 5th IMA Conference, number 1025 in Lecture Notes in Computer Science*, pp. 100–111, Berlin, 1995.

[10] F. J. MacWilliams and N. J. A. Sloane, "The theory of error-correcting codes," pp. 526–527, North-Holland, 1979.