Restricted Clustered Neural Network for Storing Real Data

Robin Danilo
Université de Bretagne Sud,
Lab-STICC
Lorient, France
robin.danilo@univ-ubs.fr

Vincent Gripon
Electronics Department,
Télécom Bretagne
Brest, France
vincent.gripon@telecom-bretagne.eu

Philippe Coussy
Université de Bretagne Sud,
Lab-STICC
Lorient, France
philippe.coussy@univ-ubs.fr

Laura Conde-Canencia
Université de Bretagne Sud,
Lab-STICC
Lorient, France
laura.conde-canencia@univ-ubs.fr

Warren J. Gross
Department of Electrical and
Computer Engineering, McGill
University
Montreal, Québec, Canada
warren.gross@mcgill.ca

ABSTRACT

Associative memories are an alternative to classical indexed memories that are capable of retrieving a message previously stored when an incomplete version of this message is presented. Recently a new model of associative memory based on binary neurons and binary links has been proposed. This model named Clustered Neural Network (CNN) offers large storage diversity (number of messages stored) and fast message retrieval when implemented in hardware. The performance of this model drops when the stored message distribution is non-uniform. In this paper, we enhance the CNN model to support non-uniform message distribution by adding features of Restricted Boltzmann Machines. In addition, we present a fully parallel hardware design of the model. The proposed implementation multiplies the performance (diversity) of Clustered Neural Networks by a factor of 3 with an increase of complexity of 40%.

1. INTRODUCTION

In conventional computer architectures, data storage is performed by using indexed memories which require explicit addresses to access contents. On the other hand, associative memories allow retrieval of content from partial or noisy version of input data. They are thus suitable for tasks such as nearest neighbor search, mapping and set implementations, and data intrusion systems.

Recently a new model of associative network (parallel implementation of associative memory) known as Clustered Neural Networks (CNNs) has been proposed [4, 5]. This network is composed of a fully interconnected neuron layer. When the distribution of messages is uniform, this model offers a large storing diversity (number of messages stored) with a good message retrieval ability (probability to retrieve a message). Based on binary units, binary connections and a simple decoding algorithm, this associative network model allows efficient fully-parallel hardware implementations [7, 8, 3]. However, like other models of associative networks [9], diversity strongly depends on the distribution of stored messages. Recent work [2] proposed specific strategies to handle non-uniform distributions. However, these strategies imply to add material that increases the hardware complexity of the architecture.

In this paper, we propose a new approach and its fully-parallel hardware implementation to handle non-uniform distributions in CNN. Our method is similar to Restricted Boltzmann Machines [11, 6]: a hidden layer is added to the network and only the connections between the hidden layer and the input layer are permitted. The resulting network is named Restricted Clustered Neural Network (R-CNN).

The paper is organized as follow: In section II, CNN and non-uniform distributions are briefly discussed. In section III, the proposed model and its fully hardware implementation are presented. In section IV, R-CNN is simulated and compared with CNN. Section V concludes the paper.

2. CLUSTERED NEURAL NETWORK

To store messages composed of c symbols over an alphabet of size ℓ, CNN consists of using a binary neural network composed of c parts (named clusters), each containing ℓ neurons. They are capable of retrieving any of them when some of the symbols are missing.

We denote by $\mathcal{M}$ the set of messages to store. Each message m is composed of c symbols $m_0, \dots, m_{c-1}$. A partially erased version of m is denoted with $\tilde{m}$. Thus $\tilde{m}$ is such that either $\tilde{m}_i = m_i$ or $\tilde{m}_i$ is unknown.

2.1 Principles

In the rest of this paper, n_{ij} designates the neuron of the cluster i associated with the j-th symbol of the alphabet and v_{ij} its value, $v_{ij} = 1$ means that n_{ij} is active while $v_{ij} = 0$ means that n_{ij} is inactive. Each neuron can be connected by a binary link to any other neuron in another cluster. These binary links are stored in the adjacency matrix $W_A(\mathcal{M})$. By

GLSVLSI'15, May 20–22, 2015, Pittsburgh, PA, USA.
Copyright © 2015 ACM 978-1-4503-3474-7/15/05 ...$15.00.
http://dx.doi.org/10.1145/2742060.2743767.

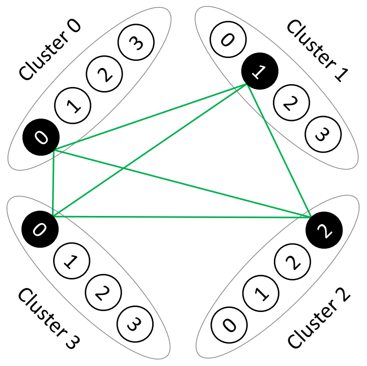

Figure 1: Graphical representation of CNN during the storing process of the message $m = (0, 1, 3, 0)$

convention $w_{(ij)(i'j')} = 1$ means that it exists a binary link between n_{ij} and $n_{i'j'}$.

During the storing process, each message m leads to the activation of one neuron in each cluster in the following manner:

$$v_{ij} = \begin{cases} 1 & \text{if } m_i = j \\ 0 & \text{otherwise} \end{cases} . \quad (1)$$

The connections between the active neurons are then stored in $W_A(\mathcal{M})$. Fig. 1 depicts the graphical representation of a CNN during the storage process of the message $m = (0, 1, 3, 0)$ (here $c = 4$ and $\ell = 4$). The neurons n_{00}, n_{11}, n_{23} and n_{30} are activated and the stored connections are represented by green lines. Storing distinct messages can result in addressing multiple times the same connections, in such case, the corresponding connection is not incremented[1].

During the retrieving process, an incomplete message $\tilde{m}$ is presented to the network leading to the activation of neurons following Eq. 1. As part of the symbols of $\tilde{m}$ are erased, some clusters of the network do not have any active neuron. We call them neutral clusters. The neutral state of each cluster is stored in a vector d, if $d_i = 1$, then, the cluster i is neutral. After this initialization phase, an iterative process is performed in order to retrieve the entire message. This process exploits two important properties of the network:

1. a unique neuron should be active in each cluster of the network,
2. for one stored message, each neuron is connected with all the other neurons of the message.

From these properties, the following activation rule is deduced: a neuron is activated if it is connected with one active neuron from each non-neutral cluster. As shown in [3], this activation rule can be performed by the following boolean function:

$$v_{ij}^{t+1} = (v_{ij}^t \vee d_i^t) \wedge \bigwedge_{i' \neq i} \left(\bigvee_{j'} \left(w_{(ij)(i'j')} \wedge v_{i'j'}^t \right) \vee d_(^t i') \right) , \quad (2)$$

where $d_i^t = \overline{\bigwedge_j v_{ij}^t}$.

Note that after an iteration, several neurons may be activated in one cluster. For this reason, the process iterates until eventually only one neuron remains active in each cluster.

[1]Note that the connections in the network are binary.

Due to the structure of the network, the memory cost of storing $W_A(\mathcal{M})$ is given by:

$$cost_{CNN} = \frac{c(c-1)\ell^2}{2}. \quad (3)$$

The performance of CNNs are evaluated in terms of diversity and message retrieval ability. The diversity is the number of messages that are learned in the network. The message retrieval ability is the probability to retrieve m from $\tilde{m}$.

2.2 Non-Uniform Distribution

CNNs offer their maximal diversity when messages to store are uniformly distributed. Real-world data lead to a significantly lower diversity. Indeed, a non-uniform distribution results in a more frequent occurrence of some symbols and thus an overuse of some neurons in a cluster.

In a CNN, symbols are directly connected together, amplifying the adverse effects of the distribution of these messages. Indeed, an overused neuron will be connected with another overused neuron and so on. This phenomenon has been well studied in [2] and some solutions have been proposed to deal with non-uniform distributions.

To illustrate the problem of non-uniform distribution we have compared the message retrieval ability of a CNN on two data sets. The first one named ”Yeast data set” [10] is composed of biological data and contains 1484 instances of 8 real attributes including two constants. We take only the 6 non-constant attributes and quantify them in 64 intervals. Each instance can be seen like a message of 6 symbols over an alphabet of size 64. The second named ”Uniform data set” is composed of 1484 random messages of 6 values between 0 and 63 uniformly distributed. For the two data sets, the resulting network is composed of 6 clusters of 64 neurons.

For the two data sets, every message (or instance) is first stored in the network. Then, one message is randomly chosen, two symbols are erased before being provided to the network that must retrieve the erased symbols. This test is repeated 10000 times and the error rate is computed. For the uniform data set, the error rate is equal to 0.3669, while for the Yeast data set the error rate is equal to 0.905.

2.3 Strategies

In [2], three strategies have been proposed. The first (Str1) consists of adding random symbols to extend the messages. These random symbols are associated with hidden clusters (which are not part of the message) to support the input clusters (which are part of the message). The network is thus composed of hidden and input clusters fully interconnected. The second strategy (Str2) is based on increasing the size of clusters instead. Thus a symbol is not associated with a unique neuron in a cluster but with a group of neurons. During the storing process, each time the symbol occurs, one of the corresponding neurons is chosen at random to be its representation. During the retrieving process, the occurrence of one symbol leads to the activation of several neurons in a cluster. The last strategy (Str3) consists of using Huffman lossless compression. The authors thus associate the most frequent symbols with a lot of corresponding neurons whereas least frequent ones are associated with few neurons.

In [2], the three strategies have been simulated for a Gaussian distribution. Str1 gives poor performance compared with the two others. For Str2 it is necessary to multiply by 4 the number of neurons in each cluster to reach the performance of the uniform distribution. As a consequence, the total amount of memory needed to store $W_A(\mathcal{M})$ is multiplied by 16. Str3 offers the best results for the same amount of memory. However, the drawback of Str3 is that it is necessary to preprocess the data before the use of CNN.

3. RESTRICTED CLUSTERED NEURAL NETWORK

In this section we propose a new approach deal with non-uniform distribution. The strategy consists in associating a hidden message randomly generated to the input message. Like the random symbol strategy, we add hidden clusters to the network for the hidden message but instead of using a fully connected CNN we only have connections between the input clusters and the hidden clusters. Thus, when a partially erased message is presented to the network, the hidden message is first retrieved and is then used to retrieve the input message. This procedure is similar to restricted Boltzmann machines and the resulting network is named Restricted CNN (R-CNN). Fig. 2 depicts a graphical representation of a R-CNN.

3.1 Principles

The number of hidden clusters c^h and the number of neurons per hidden cluster ℓ^h must be chosen by the designer depending on the application specification (intended diversity). Neurons of the input and hidden layers are noted by n_{ij}^{in} and n_{ij}^{h} respectively. In a R-CNN, a neuron of one layer can only be connected to neurons of the other layer. The connections are stored in the adjacency matrix $W_R(\mathcal{M})$, by convention, $w_{(ij)^{in}(i'j')^h} = 1$ means that a binary link exists between n_{ij}^{in} and $n_{i'j'}^{h}$.

During the storing process, each new input message m^{in} is associated with a *randomly generated* hidden message m^h. m^{in} leads to the activation of one neuron in each input cluster while m^h leads to the activation of one neuron per hidden cluster. The connections between the active neurons of the input layer and the active neurons of the hidden layer are then stored in $W_R(\mathcal{M})$. Fig. 2 depicts the graphical representation of a R-CNN during the storing process of the input message $m^{in} = (0,1,3,0)$, the hidden message $m^h = (0,2,0,3)$ is associated to the input message. The neurons n_{00}^{in}, n_{11}^{in}, n_{23}^{in}, n_{30}^{in}, n_{00}^{h}, n_{12}^{h}, n_{21}^{h}, n_{33}^{h} are activated and the stored connections are represented by green lines.

During the retrieving process, an incomplete input message $\tilde{m}^{in}$ activates neurons in the input layer. A part of the clusters of the input layer are neutral due to the erased symbols of $\tilde{m}^{in}$. Then an iterative process is performed in order to retrieve the entire message. This process is divided in two steps:

1. activation of the neurons of the hidden layer from the input layer

2. activation of the neurons of the input layer from the hidden layer

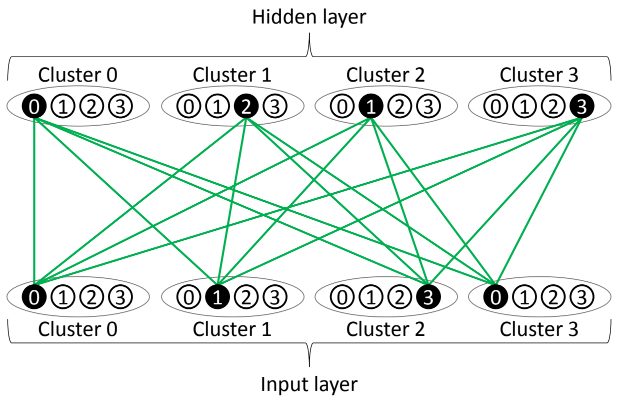

Figure 2: Graphical representation of R-CNN during the storing process of the message $m = (0,1,3,0)$

The neurons are activated by the following boolean functions:

$$v_{ij}^{h\ t+1} = \bigwedge_{i'} \left(\bigvee_{j'} \left(w_{(i'j')^{in}(ij)^h} \wedge v_{i'j'}^{in\ t} \right) \vee d_{i'}^{in\ t} \right) , \quad (4)$$

where $d_{i'}^{in\ t} = \overline{\bigwedge_{j'} v_{i'j'}^{in\ t}}$.

$$v_{ij}^{in\ t+1} = \bigwedge_{i'} \left(\bigvee_{j'} \left(w_{(ij)^{in}(i'j')^h} \wedge v_{i'j'}^{h\ t} \right) \right) , \quad (5)$$

Note that after the first step, each hidden cluster has at least one neuron activated and cannot be neutral.

Due to the structure of the network, the memory cost of storing $W_R(\mathcal{M})$ is given by:

$$cost_{R-CNN} = c.\ell.c^h.\ell^h. \quad (6)$$

3.2 Proposed Architecture

A fully parallel hardware implementation of R-CNN is proposed in this section. First, the generation of random messages for the hidden layer is explained. Then, the storing and activation modules are presented.

3.2.1 *Random Message Generation*

The messages generated for the hidden layer must be uniformly distributed. For that, the low-complexity simple algorithm proposed in [1] is used. With this algorithm, instead of dividing the set of neurons of the hidden layer into equal clusters, the size of each cluster is chosen to be relatively prime with each other. Relatively prime means that the greatest common divisor is equal to one. The size of the cluster i is ℓ_i^h and the value assigned to this cluster is m_i^h. During initialization, the value 0 is assigned to each cluster, thus the first message generated for the hidden layer is a zeros vector. Then, a new message is generated by assigning to each cluster the value computed from:

$$m_i^{h\ t+1} = (m_i^{h\ t} + 1) \pmod{\ell_i^h}. \quad (7)$$

Tab. 1 shows an example in which the hidden layer is divided in 3 clusters of sizes $\ell_0^h = 3$, $\ell_1^h = 4$, $\ell_2^h = 5$. Each symbol is represented in a cell under its integer form on the first line and under its equivalent one-hot encoded form on the second line. The interests of such algorithm are:

ℓ_i^h	3	4	5
	m_0^h	m_1^h	m_2^h
m^h 0	0	0	0
	0 0 1	0 0 0 1	0 0 0 0 1
m^h 1	1	1	1
	0 1 0	0 0 1 0	0 0 0 1 0
m^h 2	2	2	2
	1 0 0	0 1 0 0	0 0 1 0 0
m^h 3	0	3	3
	0 0 1	1 0 0 0	0 1 0 0 0
m^h 4	1	0	4
	0 1 0	0 0 0 1	1 0 0 0 0
m^h 5	2	1	0
	1 0 0	0 0 1 0	0 0 0 0 1
m^h 6	0	2	1
	0 0 1	0 1 0 0	0 0 0 1 0
m^h 7	1	3	2
	0 0 1	1 0 0 0	0 0 1 0 0
m^h 8	2	0	3
	0 1 0	0 0 0 1	0 1 0 0 0
m^h 9	0	1	4
	1 0 0	0 0 1 0	1 0 0 0 0
m^h 10	1	2	0
	0 0 1	0 1 0 0	0 0 0 0 1
m^h 11	2	3	1
	0 1 0	1 0 0 0	0 0 0 1 0

Table 1: Exemple of messages generated for the hidden layer

1. it automatically produces uniformly distributed messages,
2. it maximizes the Hamming distance between the messages produced,
3. it is easy to implement by using shift registers.

The design of the hidden layer is done as follow: first the number of hidden neurons N^h and the number of hidden clusters c^h are chosen, then, the size ℓ_i^h of each hidden cluster is determined. To determine the size of hidden clusters the mean size is first computed $\ell_{mean}^h = \frac{N^h}{c^h}$ and $c^h - 1$ prime numbers are chosen around ℓ_{mean}^h to be the sizes of the first $c^h - 1$ hidden clusters. The last size is equal to N^h minus the sum of the first $c^h - 1$ sizes. For example, for $N^h = 1536$ and $c^h = 6$, $\ell_{mean}^h = 256$, the $c^h - 1$ first sizes could be $\ell_0^h = 241$, $\ell_1^h = 251$, $\ell_2^h = 257$, $\ell_3^h = 263$, $\ell_4^h = 269$ and the last size should be $\ell_5^h = 1536 - 241 - 251 - 257 - 263 - 269 = 255$. This procedure guarantees that each size is relatively prime with each other.

3.2.2 *System Level Architecture*

Fig. 3 depicts the system level architecture of a fully parallel hardware implementation of the R-CNN. The system is composed of two neuron layers (input, hidden) each one divided in c and c^h clusters respectively. If the input clusters are of equal sizes, each hidden cluster i has a specific size ℓ_i^h in order to generate random messages uniformly distributed. During both storing and retrieving process, messages of length $\kappa.c$ ($\kappa = log_2(\ell)$) are cut into c symbols of length κ and provided to the Local Maping Modules (LMMs). LMMs realize a one hot encoding of symbols into vectors of ℓ bits where only one bit is set to one. These c vectors of ℓ bits are then stored in the neurons state registers of the input layer.

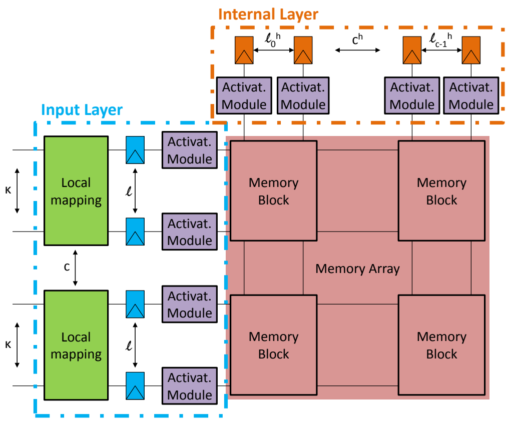

Figure 3: Simplified scheme of the system level architecture

During the storing process, a set of messages is stored in the network by storing the connections between the input layer and the hidden layer in a memory array. This memory array is divided in $c.c^h$ memory blocks composed of massively parallel on-chip registers (flip-flops) for simultaneous access. One block is used to store the connections between an input cluster and a hidden cluster. This operation is done by using the storing module. During the retrieving process, incomplete messages are provided to the system and the stored connections are used to retrieve the erased symbols by alternating between the computation of the values of the hidden state registers and the input state registers. This operation is done by using several activation modules (one per neuron).

3.2.3 *Storing Module*

During the storing process, each input message is associated with a hidden message generated with the algorithm proposed in [1]. To generate these hidden messages, each hidden cluster i is provided with a shift register whose size corresponds to the size ℓ_i^h of the cluster. For the first hidden message, each shift register has one register set to one and the others to zero. After the storage of each new input message, the bit set to one is shifted.

Fig. 4 depicts the storing module for one memory block between an input cluster and the i^{th} hidden cluster. The input cluster is composed of a set of ℓ neurons state registers while the hidden cluster is composed of a shift register of size ℓ_i^h. Between them, a memory block of size $\ell \times \ell_i^h$, where each row stores the connections of one neuron of the input cluster with all neurons of the hidden cluster. During the storing of an input message, the row of the active neuron of the input cluster is selected with the MUX, and an OR array is used to accumulate the value of the shift register with the previous connections.

3.2.4 *Activation Module*

During the retrieving process, the activation modules are used to compute the values of every state registers of the input and the hidden layers. Fig. 5 depicts an activation module for the hidden layer (left side) and for the input layer (right side). This activation module implement Eq. 4 and 5.

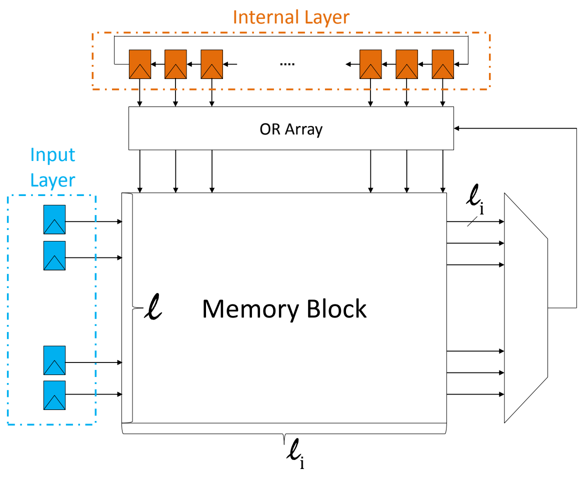

Figure 4: Block diagram of the storing module between a cluster of the input layer and a cluster of size ℓ_i^h of the internal layer

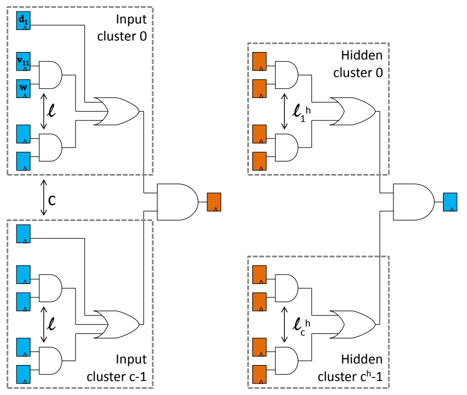

Figure 5: Logic diagram of the activation rule, for the internal layer on the left and the input layer on the right

4. RESULTS

In this section we evaluate the memory cost and the retrieval ability of R-CNN for a given diversity. A comparison is done with CNN+Str2 from [2] that is the strategy which offers the best performance without data preprocessing. Then, the proposed architecture is compared with the fully hardware implementation of CNN proposed in [8] and based on Eq. 2.

4.1 Message Retrieving

The ”Yeast data set” is used to test the two strategies, R-CNN and CNN+Str2. The 6 non-constants attributes are taken and quantified on 64 intervals. Each instance is thus a message of 6 symbols over an alphabet of size 64. The entire data set is first learned in the network, then one message (or instance) m is chosen at random, two symbols are erased to produce $\tilde{m}$ which is sent to the network in order to retrieve m. This operation is repeated 10000 times and the message retrieval ability is evaluated by computing the error rate (number of successful message retrieval over the total number of trial).

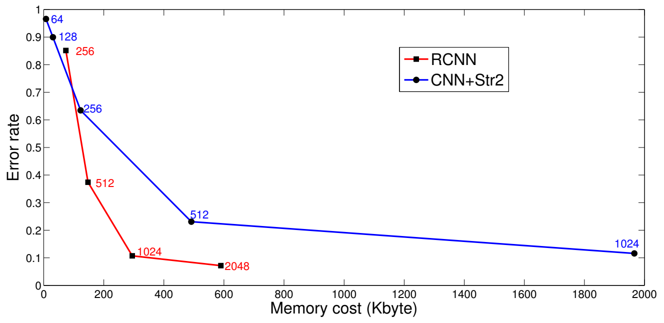

Figure 6: Error rate as a function of memory cost for R-CNN and CNN+Str2

There are several ways to increase the message retrieval ability of R-CNN: increase the size of the hidden clusters, increase the number of hidden clusters, use Str2 on the input layer. From our experiments, the greatest diversity is obtained when the size of hidden clusters is increased and we only show results for this solution.

Fig. 6 shows the error rate as a function of the memory cost for several configurations of CNN+Str2 and R-CNN. The memory cost is computed for CNN+Str2 and R-CNN with Eq.s 3 and 6 respectively. For R-CNN, the number of hidden clusters c^h are fixed to 6 and the network is simulated for different numbers of neurons in the hidden layer. The mean size ℓ^h_{mean} of the hidden clusters is reported on the figure. For CNN+Str2, different sizes of clusters are simulated, these sizes are reported on the figure. We can see in Fig. 6 that R-CNN offers better message retrieval ability than CNN+Str2 for a lower memory cost.

4.2 Complexity Analysis

We know compare the area complexity of the proposed architecture with the CNN implementation proposed in [8]. This comparison is done in NAND gate eq. (based on STMicroelectronics 90nm cell library). Two equivalent configurations of networks are chosen: a CNN with $c = 6$, $\ell = 64$ and a R-CNN with $c = c^h = 6$ and $\ell = \ell^h_{mean} = 64$. Both networks need 4 cycles to retrieve a message.

Tab. 2 presents the area cost for the two implementations. The total cost divided in three parts: the cost of registers used for memory array and neurons, the cost of storing module and the cost of retrieving module. For the CNN implementation, the major part of the area (46%) is used for registers, then the storing module (33%) and finally the retrieving module (21%). The high register cost is due to the fact that the implementation [3] uses two times more memory elements for the memory array than needed (from Eq. 3). The R-CNN implementation uses globally a greater area than the CNN implementation (+43% for the total area). The largest increase comes from the retrieving module (+127%). This is due to the number of neurons two times higher in a R-CNN than in a CNN.

If both networks give the same performance for uniform distribution, the area overhead paid for R-CNN is compensated by better performance for non-uniform distribution. With the CNN configuration studied here, only 25 instances of the ”Yeast data set” can be stored with an error rate under 0.01, whereas the number of instances which can be stored in

	CNN [3]	R-CNN
Registers	1.36×10^6	1.63×10^6 (+20%)
Storing	0.97×10^6	1.17×10^6 (+20%)
Retrieving	0.62×10^6	1.41×10^6 (+127%)
Total	2.95×10^6	4.01×10^6 (+43%)

Table 2: Area of CNN ($c = 6$ and $\ell = 64$) and R-CNN ($c = c^h = 6$ and $\ell = \ell^h_{mean} = 64$) in NAND gate eq.

the R-CNN for the same error rate is equal to 76 (multiplied by a factor of 3).

5. CONCLUSION

In this paper, a new model of associative memory named the Restricted Clustered Neural Network (R-CNN) has been introduced. This model is based on the Clustered Neural Network (CNN) and uses Restricted Boltzmann Machine principles in order to increase the diversity of CNN on non-uniform distributions of input messages. A comparison was done between the proposed model and the CNN in terms of memory usage and error rate for a message retrieval task on real data (non-uniform distribution). It has been shown that the R-CNN outperforms CNN for this task with a lower memory usage.

A fully parallel implementation of R-CNN has also been proposed. This implementation has been compared with an equivalent implementation of a basic CNN. Although the area cost is more important for R-CNN, the additional cost (+43%) is largely compensated by better performance (diversity is multiplied by a factor of 3).

The storing algorithm proposed here is simple and easy to implement in hardware. It consists of associating a hidden message uniformly distributed with the input message (non-uniformly distributed). In future work, we will study smarter storing algorithms which choose the hidden message function of input messages.

Acknowledgment

This work has received a French state support granted to the CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Fuure" program under reference Nb. ANR-10-LABX-07-01.

6. REFERENCES

[1] E. B. Baum, J. Moody, and F. Wilczek. Internal representations for associative memory. *Biological Cybernetics*, 59(4-5):217–228, 1988.

[2] B. Boguslawski, V. Gripon, F. Seguin, and F. Heitzmann. Huffman coding for storing non-uniformly distributed messages in networks of neural cliques. In *AAAI 2014: the 28th Conference on Artificial Intelligence*, volume 1, pages 262–268, 2014.

[3] R. Danilo, H. Jarollahi, V. Gripon, L. Conde-Canencia, P. Coussy, and W. J. Gross. Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*. IEEE, 2015. to appear.

[4] V. Gripon and C. Berrou. A simple and efficient way to store many messages using neural cliques. In *Proceedings of IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain*, pages 54–58, Paris, France, April 2011.

[5] V. Gripon and C. Berrou. Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, 22(7):1087–1096, July 2011.

[6] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[7] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross. Architecture and implementation of an associative memory using sparse clustered networks. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 2901–2904. IEEE, 2012.

[8] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross. Reduced-complexity binary-weight-coded associative memories. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 2523–2527. IEEE, 2013.

[9] A. Knoblauch, G. Palm, and F. T. Sommer. Memory capacities for synaptic and structural plasticity. *Neural Computation*, 22(2):289–341, 2010.

[10] M. Lichman. UCI machine learning repository, 2013.

[11] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.