REDUCED-COMPLEXITY BINARY-WEIGHT-CODED ASSOCIATIVE MEMORIES

Hooman Jarollahi, Naoya Onizawa, Vincent Gripon, and Warren J. Gross

Department of Electrical and Computer Engineering, McGill University, Montreal, Quebec, H3A 0E9
{hooman.jarollahi, naoya.onizawa, vincent.gripon} @mail.mcgill.ca, warren.gross@mcgill.ca

ABSTRACT

Associative memories retrieve stored information given partial or erroneous input patterns. Recently, a new family of associative memories based on Clustered-Neural-Networks (CNNs) was introduced that can store many more messages than classical Hopfield-Neural Networks (HNNs). In this paper, we propose hardware architectures of such memories for partial or erroneous inputs. The proposed architectures eliminate winner-take-all modules and thus reduce the hardware complexity by consuming 65% fewer FPGA lookup tables and increase the operating frequency by approximately 1.9 times compared to that of previous work.

Index Terms— Associative Memories, Hopfield, Clustered Neural Networks, Hardware Implementation

1. INTRODUCTION

Associative Memories (AMs) such as Hopfield Neural Networks (HNNs) [1] learn messages by linking the information bits, and storing the links in a memory module. They can then quickly retrieve the messages given only part of their content and using the stored links. Such memories are alternatives to indexed memories in which the retrieval process requires an explicit address and often a serial search operation. AMs are attractive in certain applications such as data mining and set implementation where the computations can benefit from their specific functioning [2, 3, 4, 5] where the exhaustive search operation in series is eliminated.

Content-Addressable Memories (CAM) are specific types of such memories used in a variety of applications such as image processing and network routers. However, CAMs consume large amounts of power due to the presence of parallel comparators [6]. In some applications such as Translational Lookaside Buffers (TLBs), they can only contain few entries [7].

The state-of-the-art HNNs suffer from two major drawbacks: First, in order to increase the size of memory, the length of the messages need to be unnecessarily increased. Therefore, the number of different messages HNNs can learn (diversity) is low since due to a fixed capacity, they can only store few long messages instead of many short ones. Second, the ratio between the number of information bits that it can store to the memory bits that it requires to store the links (efficiency) approaches zero as the memory size is increased.

Recently, a new class of AMs has been proposed [2] based on Clustered-Neural-Networks (CNNs) which has a large diversity, and provides a nearly-optimal efficiency [8].

A proof-of-concept hardware architecture of CNNs has recently been proposed in [9], where the authors show that when implemented in hardware, the architecture provides significant speedup compared to its software counterpart. However, the previous architecture requires resource-hungry Winner-Take-All (WTA) modules, which are based on a classical compare-and-select algorithm.

In this paper, we introduce two hardware architectures: one useful for applications that need to recover partly erased input patterns and the other to correct inputs with erroneous bits. The proposed architectures introduce new decoding methods that eliminate the requirement to employ the WTA modules in [9, 8, 2], and also improve the design complexity by reducing the area requirement and increasing the maximum operating frequency. In addition, the proposed architectures have a similar error performance – the number of correctly retrieved messages to that of the total inputs. The algorithm is briefly explained from a hardware design perspective in Section 2, and the hardware implementations are introduced in Section 3. Section 4 summarizes the results and is followed by conclusions in Section 5.

2. ASSOCIATIVE MEMORIES USING CLUSTERED NEURAL NETWORKS

Inspired from HNNs [1], CNNs are made of simple computing nodes (neurons) and binary connection weights as shown in Fig. 1. More precisely, the network consists of n binary neurons arranged into c equally-partitioned clusters. Each cluster is associated with a portion of a message to be learned or retrieved. Unlike HNNs, the connection weights in CNNs are binary, i.e. a connection either exists or not. In addition, the network is c-partite which means that two neurons in a same cluster cannot be connected. A partial input pattern is presented and a sparse set of neurons are activated which represent the matching learned pattern. The set is then encoded to form the full output pattern, also called a *clique*.

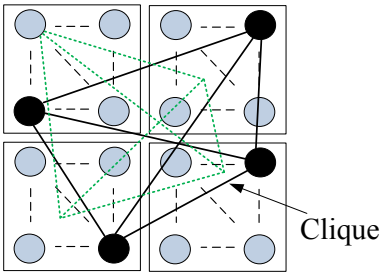

Fig. 1. Graphical representation of neurons, clusters and cliques.

2.1. Message Training

The training of the network is initiated by dividing a binary input message m of K bits into equal portions of κ bits each resulting in the creation of $c = K/\kappa$ sub-messages. Each cluster in the network consists of $l = 2^{\kappa}$ binary neurons. For simplicity, a simple function that maps the binary value of each input sub-message to its equivalent integer number between 1 to l is considered. This integer value is the index of the neuron to be activated in each cluster. Once the neurons corresponding to an input message are determined in all of the clusters, the corresponding binary connections are added to the network, and stored in a block of memory. In other words, once all the connections have been determined for the message to learn, a neural clique is constructed as shown in Fig. 1.

In this paper $w_{(c_i,l_j)(c_{i'},l_{j'})}$ refers the binary interconnection weights between the j'th neuron of the i'th cluster to the j' neuron of the i' cluster.

2.2. Message Retrieval

To retrieve a message, we first determine which neurons should be activated given a partial or erroneous input pattern and using a similar method as in training. Because we are going to match against a partial message, multiple neurons for a cluster might be activated. We then use the interconnection weights between clusters to allow the learned associations between portions of the learned message to affect which neurons should be activated. This process iterates until only one neuron per cluster is activated or the number of activated neurons is not changed. The active set of neurons is then encoded to form the output. The decoding process is performed in two stages: Local Decoding (LD) and Global Decoding (GD). Only GD is performed iteratively to retrieve the message.

2.2.1. Conventional Algorithm

Local Decoding: In the conventional LD [2, 3, 8, 9], the index of the neurons to be activated in each cluster is determined using the scalar product of a pre-designed matrix of size $l \times \kappa$ bits, g, with each part of the input message I. The g matrix contains ordered binary values between 1 to l. The process is then followed by finding the maximum value of the neurons that would exceed a certain threshold value σ. The last step is referred to winner-take-all rule in neural networks where only the neuron(s) with maximum value (v_{max}) will be activated.

Global Decoding: Following LD, GD is performed using the product of the neural values obtained from LD and the stored weight values. The winner-take-all rule is then applied. This procedure continues in an iterative process on each neuron until the values converge.

2.2.2. Proposed Local Decoding

It is possible to simplify the functioning of the conventional LD [2, 8, 9] by considering the fact that the maximum possible value for a neuron in LD is directly dependent on the number of erased bits per cluster. Therefore, it can be derived to be equal to $\kappa - n_e$ where n_e is the number of erased bits (1). This simplification will eliminate the need to apply the winner-take-all rule after finding scalar product of g and I.

$$v(n_j) \leftarrow \begin{cases} 1, & \text{if } v(n_j) = \kappa - n_e \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

2.2.3. Proposed Global Decoding

It is possible to reduce the complexity of the conventional GD using two methods:

Method I: It can be shown that the value of a neuron being globally decoded can be determined to be (or remain) activated only if it receives at least one signal from every other cluster than itself, i.e. the ambiguities of other clusters will not have an effect on the value of the neuron being computed in GD [8] as shown in the following:

$$v(n_{i,j}) \leftarrow \begin{cases} 1, & \sum\limits_{c} \bigvee_l \left(w_{(i',j')(i,j)} v(n_{(i',j')}) \right) + \\ & \gamma v(n_{(i,j)}) \geq \sigma \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $v(n_{(i,j)}$ is the binary value of the j'th neuron in the i'th cluster, $\bigvee_l$ performs an l-input OR function, and σ is a threshold value that can be adjusted to fine-tune the error rate in case the input messages are erroneous instead of containing erased bits. If $\sigma = \gamma + c - 1$, this method can only be useful for retrieving inputs containing erased bits and not for the erroneous ones.

As an example, let us assume a network comprised of three clusters ($c = 3$), two neurons each ($l = 2$), $\gamma = 1$, and a single learned message "010". An erroneous input message "110" will activate a false neuron in the first cluster. Therefore, if $\sigma = c$, all neurons will switch off after the first iteration and therefore the network will not retrieve the message. On the other hand, if $\sigma = c - 1$, the network will be able to retrieve the correct message.

This algorithm is the basis of the first proposed hardware implementation (Architecture I) explained later in this paper.

Method II: In cases where the input messages are not erroneous and only contain erased bits, a neuron is activated

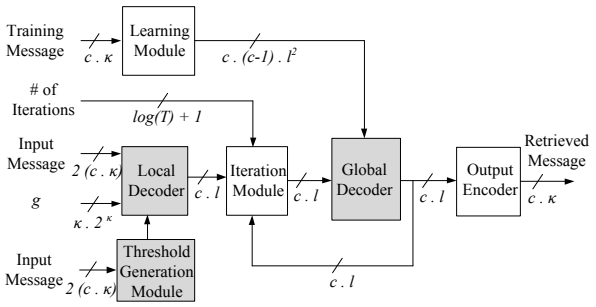

Fig. 2. Simplified block diagram of the top level hierarchy with shaded proposed modules compared to [9].

when it equal to $\gamma+c-1$, where γ is set to 1 in this paper. This simplification is the basis of the second proposed hardware implementation of this paper (Architecture II).

Method I (2) can therefore be altered to suit digital circuit implementation by removing the sum and using logic symbols as follows:

$$v(n_{i,j}) \leftarrow \begin{cases} 1, & \text{if} \quad \bigwedge_c \bigvee_l \left(w_{(i',j')(i,j)} v(n_{(i',j')})\right) \\ & \qquad \bigwedge v(n_{(i,j)}) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $\bigwedge_c$ performs the c-input AND function. This method can only be used for a case when bit erasures exist, and will not work for erroneous input corrections. This algorithm is the basis of Architecture II explained later in this paper.

3. PROPOSED HARDWARE IMPLEMENTATION

The proposed architectures are based on the proposed local decoding (1), Method I (2) and Method II (3) global decoding algorithms explained earlier. The new hardware decoding algorithms were implemented using the same network parameters (128 neurons, 8 clusters) as in [9], such that the comparison would be consistent. Furthermore, a larger network (432 neurons, 8 clusters) was also implemented to demonstrate how the ratio of logic to memory is scaled in the proposed architecture.

3.1. Design Hierarchy

A top-level system diagram addressing the proposed hardware implementations is depicted in Fig. 2. The input messages contain antipodal values (-1 or $+1$) or zeros to indicate erasure. The difference between this design hierarchy and that reported in [9] is in the way both the local and the global decoders are implemented.

3.2. Message Processing

An incoming message is either to be trained or retrieved. The training of the network is achieved by storing the binary connection weights in a two-dimensional Flip-Flop (FF) array as shown in [9]. These FF's are accessed independently. The amount of memory required to store these connection weights is given by $c(c-1)l^2$.

3.2.1. Proposed Architecture of Local Decoder

The proposed architecture of LD is based on (1). The process is initiated determining the threshold value using threshold generator as shown in Fig. 3 (a) which can vary depending on the number of erased bits in a cluster. The generated threshold values are fed into the LD module as shown in Fig. 3 (b). It activates the neuron(s) in each cluster and no longer requires resource-hungry WTA modules as in [9].

3.2.2. Proposed Architectures of Global Decoder

The proposed GD architectures are depicted in Fig. 4(a), 4 (b). These architectures are called GD architecture I and architecture II and are based on (2) and (3) respectively. To elaborate on the behaviour of these structures, let us assume that the value of the j'th neuron of cluster i is to be computed. The global decoder computing the value of each neuron has three types of inputs: 1) $w_{(i,j)(i',j')}$, the binary weights from the training module connecting all neurons excluding the i'th cluster to $n_{i,j}$, 2) $v_{(i',j')}$, the neuron's binary value in the clusters excluding the i'th cluster, and 3) $v_{(i,j)}$, the neuron's binary value of the computing neuron constructing the memory effect. t and $t+1$ denote the current and next iteration moments respectively. In the hardware implementation, the memory effect coefficient, γ, is considered to be equal to 1 (also discussed in [2, 9]), which will also simplify the hardware design since all neuron values will be integers in all iterations.

In GD architecture I, similar to the previous architecture in [9], the multiplication of the binary weights, obtained from the memory, and the neural values in the clusters excluding the cluster of which the neural value is being computed, is achieved using two-input AND gates. As opposed to the previous method (finding the maximum value using the compare-and-select method), the binary multiplication results construct the inputs to a set of multiple-input OR gates which will output one signal per cluster. Then, the output from the OR gates, as well as the neural value obtained from LD are added together and compared with a threshold value equal to c or lower. The target neuron will remain activated if the comparator outputs a '1'. Removing the adder and replacing an c-input AND gate will result in a fixed threshold for Architecture II as shown in Fig. 4 (b).

4. RESULTS

The proposed architectures of CNNs have been implemented using an Altera Stratix IV (EP4SGX230KF40C2) Field Programmable Gate Array (FPGA). After verification of the results using a similar method described in [9], the hardware complexities, and the performances were compared. The results are summarized in Table 1. Fig. 5 depicts the FPGA

Table 1. FPGA resource allocation comparing CNN architectures with design parameters reported in [9].

	Previous work [9]	Architecture I	Architecture II
Memory usage dedicated to w (bits)	14,336	14,336	14,336
Registers	$15,783/182,400(9\%)$	$15,048/182,400(8\%)$	$15,035/182,400(8\%)$
Combinational Look-up Tables (LUT)	$35,224/182,400(19\%)$	$13,244/182,400(7\%)$	$12,341/182,400(7\%)$
Total pins	$169/888(19\%)$	$169/888(19\%)$	$169/888(19\%)$
Slow 900mv 85C maximum frequency (Mhz)	107.15	203.78	205.21
Training, Retrieving delay (per message)	10 ns, 50 ns	5 ns, 25 ns	5 ns, 25 ns

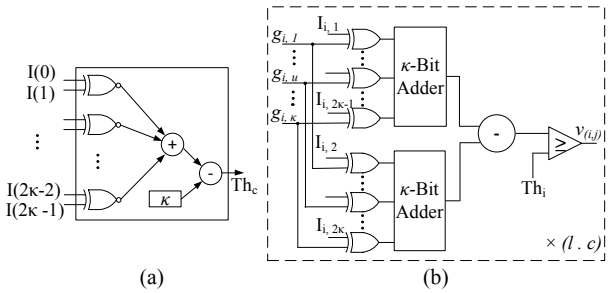

Fig. 3. Block diagram of the architecture of (a) the threshold generator (b) Local Decoder.

Table 2. FPGA resource allocation comparing two networks ($n = 128$) and ($n = 432$).

	$n = 128$	$n = 432$
Registers (Prev.[9])	15, 783	160, 745
Registers (Proposed Arch. II)	15, 035	161, 331
LUT (Prev.[9])	35, 224	359, 127
LUT (Proposed Arch. II)	12, 341	147, 808

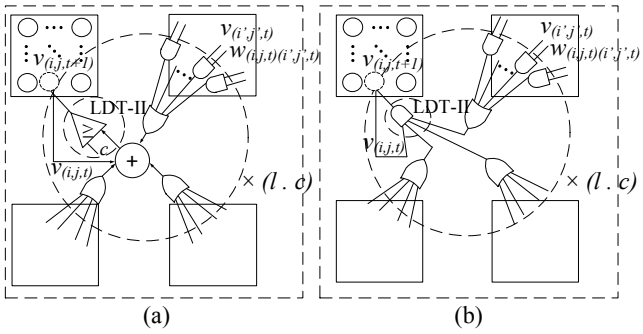

Fig. 4. Hardware implementation of the proposed Global Decoder Architecture I (a) and II (b).

results of error performance for various densities. Each point obtained from the FPGA is associated with the average MER of 1000 test vectors applied to each of the 20 networks under test. It also shows software simulation results with similar parameters. The message densities are calculated according to [2].

Using a network with $n_1 = 128$ neurons and 8 clusters, Architecture I and II consume 62.4% and 65.0% fewer LUTs respectively than Architecture I while the number of registers are similar. With a larger network comprised of $n_2 = 432$ neurons and 8 clusters, the previous architecture, unlike the proposed ones in this paper, can no longer fit within the same area as shown in Table 2. Furthermore, it is interesting to note that $n_2/n_1 \approx 3.4$ is larger than the ratio of the corresponding number of LUTs (≈ 2.43). This comparison shows that growth of the number of neurons is larger than the growth of the number of LUTs. The maximum clock frequency in the FPGA has also been improved by ≈ 1.9 times resulting in lower computational delay.

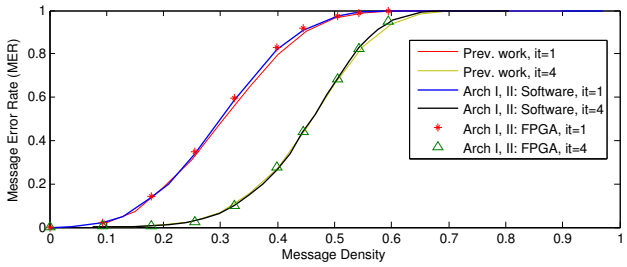

Fig. 5. Comparison between the average MERs of the proposed architectures and that of [9] using FPGAs, and software simulation ($n = 128, c = 8$, Number of erased clusters (random position): 4, FPGA number of input test vectors per network: 1000, FPGA number of networks per density: 20).

5. CONCLUSION

In this paper two hardware architectures were proposed for a new class of associative memories. The proposed architectures are suitable to either recover partially erased input patterns, or to correct erroneous input bits respectively. These memories can be used in a variety of applications such as data mining, and set implementation. The proposed architectures introduce decoding methods that reduce the hardware complexity by consuming 62.4% and 65% fewer lookup tables respectively. Furthermore, the proposed architectures increase the operating frequency approximately 1.9 times compared to that of previous work while having a similar error performance. We introduced novel hardware techniques that eliminated the resource-hungry Winner-Take-All circuits in the previous architecture, and replaced it with simpler logic without sacrificing the error performance. The results were compared with that of previous work and their functionalities were verified using a similar verification strategy.

6. REFERENCES

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the USA*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[2] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *IEEE Transactions on Neural Networks*, vol. 22, no. 7, pp. 1087–1096, Jul. 2011.

[3] V. Gripon and C. Berrou, "A simple and efficient way to store many messages using neural cliques," in *2011 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*, Apr. 2011, pp. 1–5.

[4] A. Knoblauch, "Optimal synaptic learning in non-linear associative memory," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2010, pp. 1–7.

[5] O. Qadir, J. Liu, J. Timmis, G. Tempesti, and A. Tyrrell, "Hardware architecture for a bidirectional hetero-associative protein processing associative memory," in *2011 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2011, pp. 208–215.

[6] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[7] Yen-Jen Chang and Mao-Feng Lan, "Two new techniques integrated for energy-efficient TLB design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 1, pp. 13–23, Jan. 2007.

[8] Vincent Gripon and Claude Berrou, "Nearly-optimal associative memories based on distributed constant weight codes," in *Proceedings of Information Theory and Applications Workshop*, San Diego, CA, USA, Feb. 2012, pp. 269–273.

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