A GPU-based Associative Memory using Sparse Neural Networks

Zhe Yao
Electrical and Computer Engineering
McGill University
Montréal, Canada
zhe.yao@mail.mcgill.ca

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

Michael Rabbat
Electrical and Computer Engineering
McGill University
Montréal, Canada
michael.rabbat@mcgill.ca

***Abstract*—Associative memories, serving as building blocks for a variety of algorithms, store content in such a way that it can be later retrieved by probing the memory with a small portion of it, rather than with an address as in more traditional memories. Recently, Gripon and Berrou have introduced a novel construction which builds on ideas from the theory of error correcting codes, greatly outperforming the celebrated Hopfield Neural Networks in terms of the number of stored messages per neuron and the number of stored bits per synapse. The work of Gripon and Berrou proposes two retrieval rules, SUM-OF-SUM and SUM-OF-MAX. In this paper, we implement both rules on a general purpose graphical processing unit (GPU). SUM-OF-SUM uses only matrix-vector multiplication and is easily implemented on the GPU, whereas SUM-OF-MAX, which involves non-linear operations, is much less straightforward to fulfill. However, SUM-OF-MAX gives significantly better retrieval error rates. We propose a hybrid scheme tailored for implementation on a GPU which achieves a 880-fold speedup without sacrificing any accuracy.**

***Keywords*—Associative memory, Recurrent neural networks, Parallel processing, Sparse coding, CUDA, GPGPU**

I. Introduction

The recent work of Gripon and Berrou [1], [2] proposes a new family of sparse neural network architectures for associative memories. We refer to these as *Gripon-Berrou neural networks* (GBNNs). In short, GBNNs are a variant of the Willshaw networks [3], [4] with a C-partite structure, sharing spirit with the model proposed by Moopenn et al. [5]. However, GBNNs combine the notion of recurrence from Hopfield networks [6], [7] with ideas from the field of error correcting codes and achieve nearly optimal retrieval performance. A detailed description of the GBNN architecture and operation is given in Section II.

In [8], Berrou and Gripon successfully introduce Walsh-Hadamard codes into bidirectional associative memories. The same authors also consider the use of sparse coding in a Hopfield network. They show that, given the same amount of storage, GBNNs outperform conventional Hopfield networks in diversity (the number of patterns that the network can store), capacity (the maximum amount of stored information in bits), and efficiency (the ratio between capacity and the amount of information in bits consumed by the network when diversity reaches its maximum), while decreasing the retrieval error. In [9], GBNNs are interpreted using the formalism of error correcting codes, and a new retrieval rule, SUM-OF-MAX, is introduced to further decrease the error rate. Jiang et al. [10] modify the GBNN structure to store long sequences by incorporating directed edges into the network. Aliabadi et al. [11] make the extension to store sparse messages where messages with different lengths are stored and retrieved in the same network.

To be useful in applications, it is also essential to develop fast and efficient implementations of GBNNs. Jarollahi et al. [12] demonstrate a proof-of-concept using the *field programmable gate array* (FPGA). Due to hardware limitations, their implementation is constrained to have at most 400 neurons. Larras et al. [13] implement an analog version of the network which consumes $1165\times$ less energy and is $2\times$ more efficient both in the surface of the circuit and speed, compared with an equivalent digital circuit. However, the network size is further constrained to 208 neurons in total.

This paper demonstrates the first implementation of GBNNs on a GPU using the *compute unified device architecture* (CUDA) for both retrieval rules, SUM-OF-SUM [2] and SUM-OF-MAX [9]. SUM-OF-SUM is easier to implement, because it requires only matrix-vector multiplications, a highly optimized operation in CUDA. SUM-OF-MAX is much more difficult because it involves non-linear operations, but it gives superior retrieval performance (lower error rates). Our massively parallel implementation supports a much larger number of neurons than existing ones, and is $880\times$ faster than a CPU implementation using optimized C++ libraries for linear algebra operations, without any loss of retrieval accuracy.

The tremendous speedup comes from: a) GPU's highly parallel and efficient architecture; and b) our joint retrieval scheme using aspects of both SUM-OF-SUM and SUM-OF-MAX, so that impossible neurons are eliminated early in the retrieval process and do not waste the computing resource. Although we discuss a GPU implementation in particular, we believe the ideas presented here is general enough so that it is also useful when we extend the network to other parallel architectures.

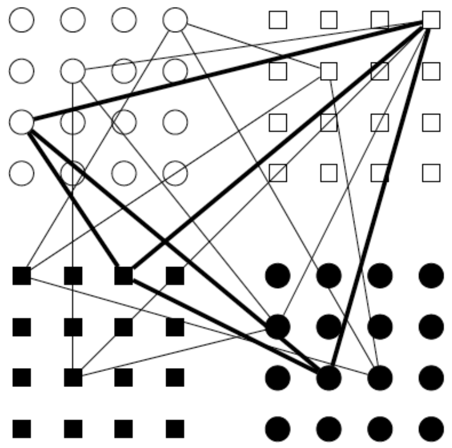


Figure 1. An example of a network with 4 clusters of 16 neurons each [2]. We number the clusters from left to right and from top to bottom as $1 \cdots 4$. The same scheme applies for neurons $1 \cdots 16$ within each cluster.

II. Gripon-Berrou Neural Networks (GBNNs)

The architecture of a GBNN [2] is closely coupled with the structure of the stored messages. Suppose a message M can be divided into a tuple of C symbols, $M = (m_1, m_2, \dots, m_C)$, where each symbol m_c can take L possible values, from x_1 to x_L. Then the network comprises C clusters of L binary-valued (0 or 1) neurons each. A message $M = (m_1, \dots, m_C)$ is represented by activating the only neuron corresponding to the value of m_c in each cluster, and setting all other neurons to 0. In this way, the message is naturally encoded as a binary string of length $n = CL$ with exactly C ones.

When a network is initialized, no edge exists between neurons. When storing a message, edges are added to connect all pairs of neurons which are activated for this particular message. For example, consider the network in Fig. 1, where each message contains $C = 4$ symbols and each symbol takes one of $L = 16$ possible values. The message indicated by the bold edges is (x_9, x_4, x_3, x_{10}). The edges corresponding to any single message thus associate with a *clique* (complete sub-graph).

For retrieval, the network is probed with an incomplete message, e.g., $(m_1, m_2, ?, ?)$, and it must find a stored message which completes the probe. If the network is queried with an entire message as a probe, then the problem boils down to deciding whether or not this message has been stored. For this case, it has been shown that the missed detection rate is zero, and the false positive rate depends on the number of messages stored in the network [2], [14]. GBNNs can also be used with inputs which contain errors, but the retrieval rules need to be modified accordingly.

III. Retrieval Rules

To perform a retrieval, we discuss two synchronous discrete time rules. The set of active neurons at the end of the iterative retrieval process corresponds to the network response to the probe.

A. *The* SUM-OF-SUM *Rule*

Active neurons emit signals through network connections. The simplest rule [2] consists in initializing the neurons corresponding to the missing symbols deactivated, and adding all the signals a neuron receives in the current iteration as its score. Then in each cluster, only neurons with the highest score are kept active. The scores are recalculated, and the process is repeated until a convergence criteria is met if ever.

Let $\mathrm{neuron}(c, l)$ denote the l^{th} neuron in the c^{th} cluster, and let $w_{(cl)(c'l')}$ denote an indicator variable for whether or not a connection is present between $\mathrm{neuron}(c, l)$ and $\mathrm{neuron}(c', l')$. We also denote by s^t_{cl} and v^t_{cl} respectively the score function for the number of signals $\mathrm{neuron}(c, l)$ receives and the indicator function for whether or not $\mathrm{neuron}(c, l)$ is activated at iteration t, with v^0_{cl} being the corresponding value for $\mathrm{neuron}(c, l)$ in the probe.

At the retrieval stage, the variables $w_{(cl)(c'l')}$ are fixed. As a consequence, the retrieval procedure can be formalized as

$$s^t_{cl} = \gamma v^t_{cl} + \sum_{c'=1}^{C} \sum_{l'=1}^{L} \left(v^t_{c'l'} w_{(c'l')(cl)} \right) \tag{1}$$

$$s^t_{c,\max} = \max_{1 \le l \le L} s^t_{cl} \tag{2}$$

$$v^{t+1}_{cl} = \begin{cases} 1 & \text{if } s^t_{cl} = s^t_{c,\max} \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $\gamma \ge 0$ is a reinforcement factor, which influences the extent a neuron's own value contributes to its signal at the current iteration. Essentially, (1) counts the score for each neuron. It involves summing over all clusters and all neurons within each cluster, hence the name SUM-OF-SUM. (2) finds the value of the neurons with the strongest signal in each cluster, and (3) keeps them active.

Unfortunately, SUM-OF-SUM is not consistent with the clique structure such that for a particular message, each neuron should ideally receive one signal per cluster only. This defect undermines the retrieval correctness severely, especially when erased clusters increases; see Fig. 4 in Section V. Also it is not guaranteed to converge either, even in a very simple setup [15].

B. *The* SUM-OF-MAX *Rule*

SUM-OF-MAX is proposed in [9], formally described as follows:

$$s^t_{cl} = \gamma v^t_{cl} + \sum_{c'=1}^{C} \max_{1 \le l' \le L} \left(v^t_{c'l'} w_{(c'l')(cl)} \right) \tag{4}$$

$$v^{t+1}_{cl} = \begin{cases} 1 & \text{if } \ s^t_{cl} = \gamma + C - 1 \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

(4) involves a summation over *max* operation, hence the name SUM-OF-MAX. The basic idea is that, to retrieve a correct

message, the score of a neuron should not be larger if it receives multiple signals from the same cluster. The maximum operation taken in (4) ensures each neuron receives at most one signal from each cluster. Since each stored message corresponds to a clique of C neurons, one in each cluster, a neuron should be activated if it receives *exactly* $C-1$ signals from the other clusters plus some value γ from the self loop.

For SUM-OF-MAX to work properly, the network must be initialized appropriately when a probe is presented. Instead of deactivating all neurons associated with missing symbols as in SUM-OF-SUM, we initialize them to be active. In that case, other neurons will definitely receive signals from these missing clusters, L signals per missing cluster, but they will be regulated by (5).

IV. ACCELERATIONS

A. *Accelerating* SUM-OF-SUM

Vectorization is a first step to fully utilize GPU's highly parallel architecture. If we stack all neurons in a row and reindex $\mathrm{neuron}(c,l)$ to be $\mathrm{neuron}((c-1)L+l)$, it is not difficult to visualize that (1) can be rewritten as

$$s^t = Wv^t, \tag{6}$$

with

$$W = \begin{pmatrix} \gamma & w_{12} & \cdots & w_{1n} \\ w_{21} & \gamma & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & \gamma \end{pmatrix},$$

whose element w_{ij} is the indicator if $\mathrm{neuron}(i)$ and $\mathrm{neuron}(j)$ are connected in the new index system. Thus, the score equation (6) is a matrix-vector product, computed efficiently in parallel on a GPU.

Algorithm 1 The SUM-OF-SUM retrieval procedure.

Input: the maximum number of iterations permitted t_{max}, the weight matrix W, the partially erased message vector v^0
Output: the recovered vecor v^t

```
1: t = -1
2: repeat
3:     t = t+1
4:     $s^t = Wv^t$
5:     $v^{t+1}$ = the kernel function as in Fig. 2
6: until $v^{t+1} == v^t$ or $t == t_{max}$
```

The pseudocode for the SUM-OF-SUM procedure is given in Algorithm 1. To update v^t, a dedicated thread processes one cluster (see Fig. 2), finding the maximum value in that cluster, and then keeping the neurons that reach the maximum value active. The retrieval procedure terminates when either the network converges or it reaches the maximum number of iterations permitted.

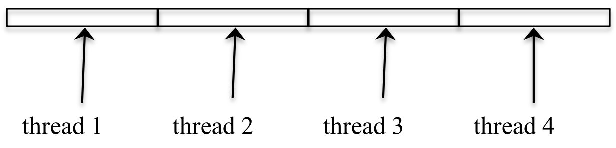


Figure 2. Illustration of the CUDA kernel function for SUM-OF-SUM. Each rectangular represents a cluster. A dedicated thread will determine the maximum value in its cluster and set the corresponding neurons active.

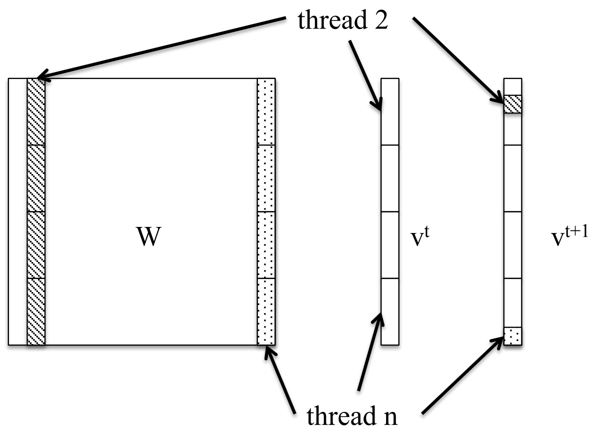


Figure 3. Illustration of the kernel function for the SUM-OF-MAX rule. To update the element v_i^{t+1}, we examine both v^t and the ith column of W.

B. *Accelerating* SUM-OF-MAX

The pseudocode for SUM-OF-MAX is almost the same as Algorithm 1, except that Lines 4 and 5 are replaced by another CUDA kernel function illustrated in Fig. 3. We do not follow strictly (4) and (5) to evaluate a *max* function. Equivalently we check if a neuron receives signals from every cluster; hence, to update $\mathrm{neuron}(i)$'s value v_i^{t+1}, a dedicated thread i is required, examining through both v^t and the ith column of W.

Thread i loops through cluster c, from 1 to C.

- For any positive γ, if $\mathrm{neuron}(i)$ belongs to cluster $c = \lfloor \frac{i-1}{L} \rfloor + 1$, we directly set $s_i^t = s_i^t + 1$. ($\lfloor \cdot \rfloor$ is the standard floor operator.)
- Otherwise, we check within the same cluster, i.e., w_{ji} and v_j^t, where j goes from $(c-1)L+1$ to cL. The first time we encounter $w_{ji} > 0$ and $v_j^t > 0$, we set $s_i^t = s_i^t + 1$, and proceed to the next cluster $c+1$ without further investigation.
- If cluster c does not contribute any signal to $\mathrm{neuron}(i)$, i.e., s_i^t does not change, we stop right away without checking following clusters.

We favor this approach over (4) and (5) for two reasons:

1) It explicitly clarifies the requirement that every cluster should contribute one and only one signal.
2) It proceeds to subsequent clusters or stops processing as quickly as possible so that further expensive memory accesses are avoided.

C. Joint Scheme

SUM-OF-MAX is much more computationally demanding than SUM-OF-SUM, because to update a single value, two vectors of length n are examined. It is mentioned in Section III that for SUM-OF-MAX to operate correctly, all neurons associated with the missing symbols need to be activated initially. However, we manage to prove in [15] that a neuron will never turn back to active in sequel iterations once deactivated. Therefore, if we can eliminate impossible neurons from the startup, it will save computations considerably.

Notice that a partial probe with e symbols erased always has $C - e$ neurons active. Therefore, a desired neuron should receive signals from those $C - e$ active neurons. We run one iteration of SUM-OF-SUM (efficient matrix-vector product) to activate those neurons before feeding them into SUM-OF-MAX, with the rest inactive during the whole retrieval process. Simulation results in Section V demonstrate the joint scheme's exciting performance.

V. Experiments

The CPU experiments are executed on a 2.6GHz AMD Phenom (tm) 9950 Quad-Core Processor with 16GB of RAM. In order to make as fair a comparison as possible, our CPU code makes full use of the Armadillo library [16], linked to BLAS and LAPACK, for optimized linear algebra operations. The GPU experiments are executed on an NVIDIA C1060 card, which runs at a frequency of 1.3GHz with 4GB memory and has 30 stream multiprocessors. The run time configuration for the kernels is 256 threads per block and 240 blocks per grid. We use the built in CUBLAS library for the matrix-vector multiplication.

Fig. 4 demonstrates the runtime (in seconds) and retrieval rate of the joint scheme compared with SUM-OF-SUM and SUM-OF-MAX for two scenarios, while varying the number of erased symbols. The spikes in runtime for SUM-OF-MAX and for the hybrid scheme in Fig. 4a are due to the fact that decoding becomes more difficult as the number of erased clusters increases, consequently more iterations are required in these cases. If too much information is missing (7 clusters erased), a great number of stored messages correspond to this same probe, the network converges quickly to the state where all neurons activates, where the retrieved pattern is useless. In Fig. 4a, although the retrieval rate is significantly lower, SUM-OF-SUM runs a bit faster when 6 clusters are erased. However, this is an illusion, since we impose a limit on the number of iterations permitted. Note that increasing this limit does not improve any retrieval rate, but it can make the runtime arbitrarily worse because SUM-OF-SUM oscillates, whereas the joint scheme and SUM-OF-MAX always converge. If the number of stored messages increases, the advantage of the joint scheme (also SUM-OF-MAX) over SUM-OF-SUM will be further pronounced. We conclude from Fig. 4 that the joint scheme combines the benefits of both existing rules, achieving fast decoding while also maintaining a high retrieval rate.

Since SUM-OF-MAX outperforms SUM-OF-SUM significantly in terms of retrieval rate, next we only consider the runtime improvements achieved by a GPU versus a CPU SUM-OF-MAX implementation. We have $C = 16$ clusters with $L = 512$ neurons each, out of which 7 clusters are erased. We generate and store 50000 random messages, with a random subset of 30000 tested. The CPU version completes the simulation in 13191.70 seconds, while our joint scheme on GPU only takes 14.86 seconds without any loss of accuracy, which is more than $880\times$ accelerated.

VI. Summary

In this work, we present optimized implementations of associative memories using Gripon-Berrou neural networks on a GPU, for both SUM-OF-SUM and SUM-OF-MAX. In order to achieve the full speedup, we combine the two rules and propose a hybrid scheme, minimizing the unnecessary computation burdens. The experimental results show an exciting acceleration against a CPU implementation using an optimized linear algebra library.

In the future, we will try to develop other retrieval schemes, e.g., to handle corrupted patterns besides incomplete probes. Since SUM-OF-SUM runs orders of magnitude faster, another sensible topic is to emulate SUM-OF-MAX using SUM-OF-SUM so that both performance and speed can be retained simultaneously. We may also seek the way to generalize the GBNN and extend the sparse neural network's use in tasks other than associative memory, e.g., classification and regression.

Acknowledgment

This work was funded, in part, by the Natural Sciences and Engineering Research Council of Canada (NSERC), the *Fonds Québécois de la recherche sur la nature et les technologies* (FQRNT) and the European Research Council project NEUCOD.

References

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

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

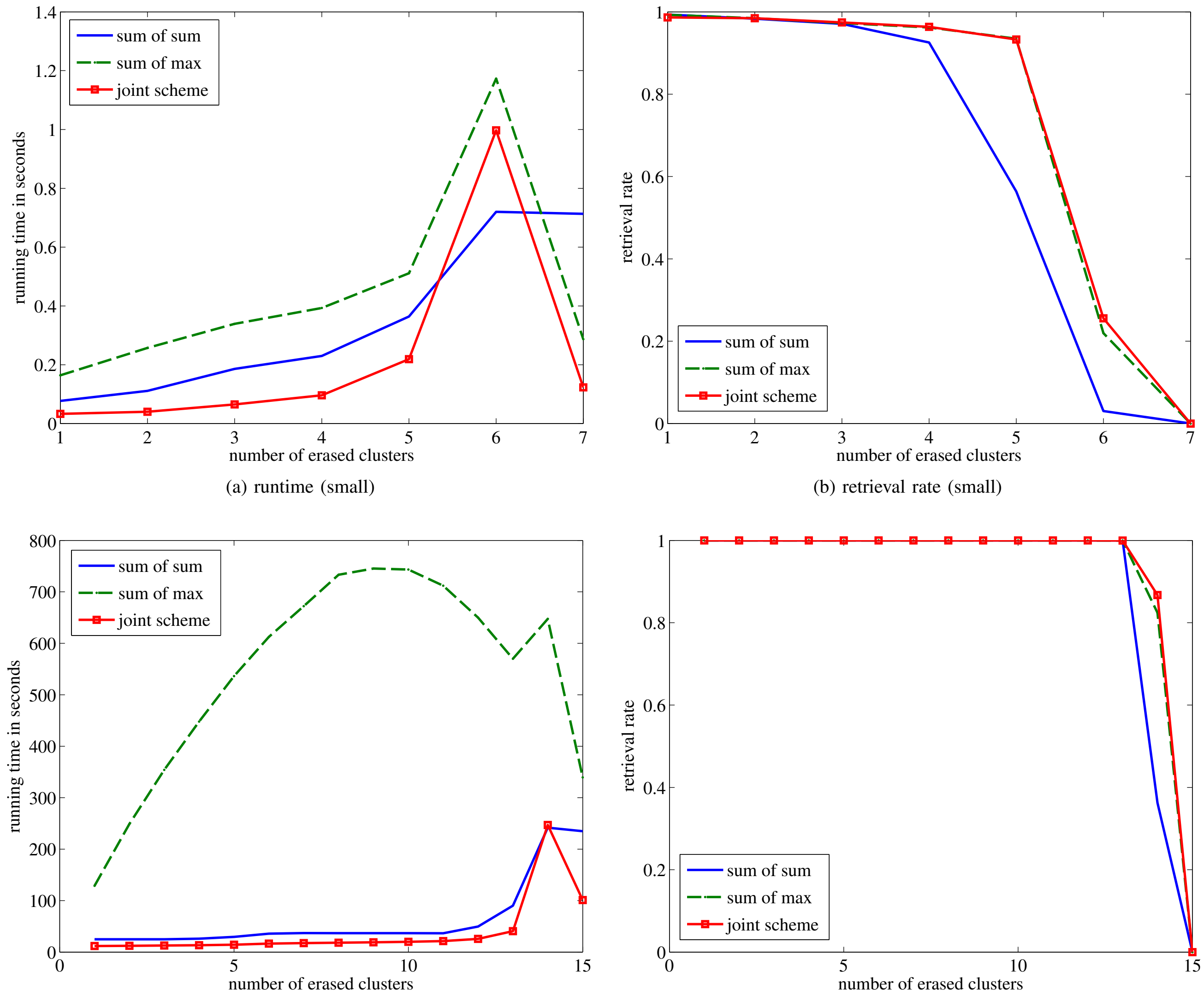
[3] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory." *Nature*, vol. 222, pp. 960–962, 1969.

[4] D. Willshaw, "Models of distributed associative memory." Ph.D. dissertation, Edinburgh University, 1971.

[5] A. Moopenn, J. Lambe, and A. Thakoor, "Electronic implementation of associative memory based on neural network models," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 17, no. 2, pp. 325–331, 1987.

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

[7] ——, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.



(a) runtime (small)

(b) retrieval rate (small)

(c) runtime (large)

(d) retrieval rate (large)

Figure 4. The behavior of the joint retrieval scheme in general. Both runtime in seconds and retrieval rate are plotted respectively as the number of erased clusters increases. We set $\gamma = 2$ and the maximum number of iterations allowed is 20. (a) and (b) refer to a small network where there are 8 clusters with 128 neurons each, 5000 messages to memorize, and 3000 messages to test. (c) and (d) refer to a large network where there are 16 clusters with 512 neurons each, 50000 messages to memorize, and 30000 messages to test.

[8] C. Berrou and V. Gripon, "Coded hopfield networks," in *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Brest, France, 2010, pp. 1–5.

[9] V. Gripon and C. Berrou, "Nearly-optimal associative memories based on distributed constant weight codes," in *Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, 2012, pp. 269–273.

[10] X. Jiang, V. Gripon, and C. Berrou, "Learning long sequences in binary neural networks," in *International Conference on Advanced Cognitive Technologies and Applications*, Nice, France, 2012, pp. 165–170.

[11] B. K. Aliabadi, C. Berrou, V. Gripon, and X. Jiang, "Learning sparse messages in networks of neural cliques," ACM Computing Research Repository, 2012. [Online]. Available: http://arxiv.org/abs/1208.4009v1

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

[13] B. Larras, C. Lahuec, M. Arzel, and F. Seguin, "Analog implementation of encoded neural networks," in *IEEE International Symposium on Circuits and Systems*, Beijing, China, 2013, pp. 1–4.

[14] V. Gripon, V. Skachek, and M. Rabbat, "Sparse structured associative memories as efficient set-membership data structures," in *Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, USA, 2013, pp. 500–505.

[15] Z. Yao, V. Gripon, and M. Rabbat, "A massively parallel associative memory based on sparse neural networks," *IEEE Transactions on Parallel and Distributed Systems*, submitted for publication. [Online]. Available: http://arxiv.org/abs/1303.7032

[16] C. Sanderson, "Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments," Technical report, NICTA, Tech. Rep., 2010.