Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Huffman Coding for Storing Non-Uniformly Distributed Messages in Networks of Neural Cliques

Bartosz Boguslawski[1], Vincent Gripon[2], Fabrice Seguin[2], Frédéric Heitzmann[1]
[1]Univ. Grenoble Alpes, F-38000 Grenoble, France
CEA, LETI, MINATEC Campus, F-38054 Grenoble, France
[2]TELECOM Bretagne, Electronics Department, 29238 Brest, France
{bartosz.boguslawski, frederic.heitzmann}@cea.fr, {vincent.gripon, fabrice.seguin}@telecom-bretagne.eu

Abstract

Associative memories are data structures that allow retrieval of previously stored messages given part of their content. They thus behave similarly to human brain's memory that is capable for instance of retrieving the end of a song given its beginning. Among different families of associative memories, sparse ones are known to provide the best efficiency (ratio of the number of bits stored to that of bits used). Nevertheless, it is well known that non-uniformity of the stored messages can lead to dramatic decrease in performance. Recently, a new family of sparse associative memories achieving almost-optimal efficiency has been proposed. Their structure induces a direct mapping between input messages and stored patterns. In this work, we show the impact of non-uniformity on the performance of this recent model and we exploit the structure of the model to introduce several strategies to allow for efficient storage of non-uniform messages. We show that a technique based on Huffman coding is the most efficient.

1 Introduction

In traditional indexed memories, data is addressed by using a known pointer. The principle of associative memories is different: data retrieval is accomplished by presenting a part (possibly small) of it. Thanks to the partial input, the rest of the information is recalled and consequently no address is needed. As a toy example, retrieving the password of a user given its name in a database is typically a request of an associative memory. Associative memories are widely used in practical applications, for instance databases (Lin, Smith, and Smith 1976), intrusion detection (Papadogiannakis, Polychronakis, and Markatos 2010), processing units' caches (Jouppi 1990) or routers (Lou and Chen 2001).

In order to assess the performance of associative memories, several parameters can be introduced. Probably the most important one (Gripon and Rabbat 2013) is termed memory efficiency and is defined as the best ratio of the total number of bits stored to the total number of bits used to store the memory itself, for a targeted performance. Note that this ratio is trivially one for indexed memories. Because of most applications, another important parameter is the computational complexity. Again, this parameter makes usually no sense for traditional memories in which computational complexity is often considered to be of $\mathcal{O}(1)$.

In practice one can distinguish two main families of associative memories, namely content-addressable memory (CAM) (Pagiamtzis and Sheikholeslami 2006) and neuroinspired memories. A CAM compares the input search word against the stored data, and returns a list of one or more addresses where the matching data word is stored. CAMs combine memory efficiency with zero error rate and are often used in electronics, for example in network routers (Pagiamtzis and Sheikholeslami 2006). However, since it is a brute-force approach, the number of comparisons between the input search word and the stored data results in high complexity and energy consumption (Jarollahi et al. 2013). Moreover, CAMs make the asumption that stored messages are couples, where only the second item can be erased in an input. Ternary CAM (TCAM) expands CAM functionality allowing "don't care" bits matching both 0 and 1 values, thereby offering more flexibility. However, this comes at an additional cost as the memory cells must encode three possible states instead of the two in case of binary CAM. Consequently, the cost of parallel search within such a memory becomes even greater (Agrawal and Sherwood 2006).

Neuroinspired associative memories combine lower complexity with higher flexibility, at the cost of reduced efficiency. In this category Hopfield Neural Networks (Hopfield 1982) is the most prominent model. In this model, stored messages are projected onto the connection weights of a complete graph. Nevertheless, when the size of this network is increased the efficiency tends to zero. Sparse networks originally proposed by Willshaw (Willshaw, Buneman, and Higgins 1969) use a small subset of connections to store each message, resulting in a much better efficiency (Palm 2013). Further, works from (Salavati and Karbasi 2012) are also known to allow for storing large number of messages.

Recently Gripon and Berrou proposed a new model (Gripon and Berrou 2011b) that can actually be seen as a particular Willshaw network with cluster structure. This modification allows for efficient retrieval algorithm without diminishing performance. This model is able to store a large number of messages and recall them, even when a significant part of the input is erased. The simulations of the net-

work working as a data structure or an associative memory proved a huge gain in performance compared to Hopfield network (Hopfield 1982) and Boltzmann machine (Ackley, Hinton, and Stejnowski 1985) (when using comparable material) (Gripon and Berrou 2011b). The fact that the network is able to retrieve messages with erasures on any position or in the presence of noise, gives it an advantage over CAMs. These interesting properties originate in error correcting codes that underlie this network principles (Gripon and Berrou 2011a).

The network as presented in (Gripon and Berrou 2011b) is analyzed only for uniform i.i.d. (independent identically distributed) values among all the messages. By the network construction this means that the number of connections going out from each node is uniformly distributed across the whole network. It is well known that non-uniformity of messages to store can lead to dramatic decrease in performance (Knoblauch, Palm, and Sommer 2010). On the other hand, it is expected that real world applications may contain highly correlated data. In this work the situation where non-uniform data is stored is analyzed and its influence on the network performance is explained. Further, we exploit the structures of these networks to introduce several techniques in order to efficiently store non-uniform data.

The paper is organized as follows. Section II outlines the theory of the network. The problem of storing non-uniform data is explained in Section III. Section IV proposes several new strategies to store non-uniform data. In Section V performance of the proposed strategies is evaluated. Section VI discusses the introduced techniques and gives an insight in the future work.

2 Sparse neural networks with large learning diversity

In this section, we introduce the family of sparse associative memories described in (Gripon and Berrou 2011b).

2.1 Message definition

Throughout this work, we consider that associative memories store *messages* that they are later capable of retrieving given a sufficiently large part of their content. In order to ease the readability of this document, and without loss of generality, we consider that a message consists of c submessages or segments. Each segment can be seen as a binary vector whose values range from 0 to $\ell - 1$. An exemplary message, for $c = 4$ and $\ell = 4$, is $m = \{10\ 00\ 01\ 11\}$ (to be read 2 0 1 3).

2.2 Network structure

In order to store messages, we use a network that consists of binary neurons and binary connections. The authors of (Gripon and Berrou 2011b), use the term *fanal* (which means lantern or beacon) instead of neuron for two reasons: a) at a given moment, in normal conditions, only one fanal within a group of them can be active and b) for biological inspirations, fanals do not represent neurons but microcolumns (Aliabadi et al. 2013). Figure 1 represents the general structure of the network and the notation. All the n fanals are

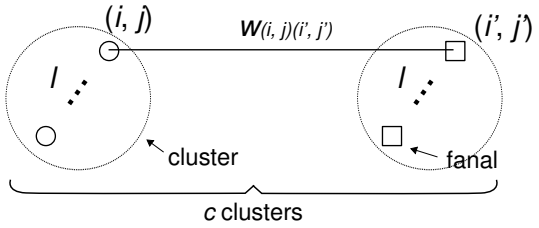

Figure 1: The network general structure and notation. Different shapes (circles, squares) represent fanals belonging to different clusters.

organized in c disjoint groups called *clusters*. Fanals belonging to specific clusters are represented with different shapes. Each cluster groups $\ell = n/c$ fanals. A node in the network is identified by its index (i, j), where i corresponds to the cluster number and j to the number of the fanal inside the cluster. The connections are allowed only between fanals belonging to different clusters, i.e. the graph is multipartite. The connection between two fanals is denoted by a binary weight $w_{(i,j)(i',j')}$. Contrary to classical neural networks the connections do not possess different weights, the connection exists or not. Hence, the weight (or adjacency) matrix of such a network consists of values $\{0, 1\}$ where 1 indicates the connection between two fanals, and 0 the lack of a connection.

2.3 Message storing procedure

To store a message m in the network, each of its c segments is associated with a distinct cluster, and more precisely with a unique fanal in its cluster (the one which index corresponds to the value of the segment). Then, this subset of fanals is fully interconnected forming a *clique* representing the message in the network. This term is also used in neurobiology to describe such groupings of neurons (Lin, Osan, and Tsien 2006). When a new message shares the same connection than an already stored message, this connection remains unchanged. Therefore, the result of the storage procedure is independent of the order in which messages are presented to the network. For more details about this storing procedure, refer to (Gripon and Berrou 2011b).

2.4 Message retrieval procedure

We call retrieval the process of recalling a previously stored message when only part of its corresponding fanals is known. After the storing of all messages, the retrieval process is organized as follows. First, the known segments of the input message are used to stimulate appropriate fanals, i.e. the value on the given segment indicates which fanal should be chosen. These fanals are said to be active. After the initial stimulation, message passing phase comes next. The activated fanals send unitary signals to other clusters through all of their connections. Then, each of the fanals calculates the sum of the signals it received. Within each cluster the fanal having the largest sum is chosen and its state becomes 1, i.e. it is active. The other fanals inside the cluster present the state equal to 0. The rule according to which the active fanal inside the cluster is elected is called *Winner*

Takes All (WTA). The whole process may be iterative, allowing the fanals to exchange information with each other, such that ambiguous clusters (those containing more than one active fanal) will hopefully be correctly retrieved. When more than one iteration is needed (input with significant noise or erasures resulting in non-unique fanal with the largest sum) an extra value is added to the score of the last winners in the next iteration. More details on adjusting this memory effect are given in (Gripon and Berrou 2011b). Hopefully, thanks to this iterative retrieval procedure the network converges step-by-step to the targeted previously stored message. In some cases though, it may happen that the output message is not correct, leading to nonzero error probability in the retrieval process.

As a result of the strong correlation brought by the connections of the clique embodying a message in the network, it is possible to retrieve the stored message based on partial or noisy information put into the network. In order to target first applications (Larras et al. 2013a), the network was implemented in hardware (Larras et al. 2013b).

3 Non-uniformly distributed messages

3.1 Density and error probability definition

As previously stated storing messages in the network corresponds to creating subgraphs of interconnected fanals. When the number of stored messages increases, these subgraphs share an increasing number of connections. Consequently, distinguishing between messages is more difficult. As a logical consequence, there is an upperbound for the number of distinct messages one can store then retrieve for a targeted maximum error probability. The network density d is defined as a ratio of the established connections to all the possible ones. Therefore, the density is a parameter of first importance to assess the network performance. A density close to 1 corresponds to an overloaded network. In this case the network will not be able to retrieve stored messages correctly. For a network that stored M uniformly distributed messages expected density d is expressed by the following formula or its first-order approximation:

$$d = 1 - \left(1 - \frac{1}{\ell^2}\right)^M \approx \frac{M}{\ell^2} \text{ when } M \ll \ell^2. \qquad (1)$$

The probability of correctly retrieving a message with c_e positions erased in a network constructed of c clusters is given by:

$$P_e = \left(1 - d^{c-c_e}\right)^{(\ell-1)c_e}. \qquad (2)$$

This equation is valid for a single iteration. The probability of error increases with d, which is expressed by (1). Note that in case of a larger density, iterations improve the ability of the network for retrieving messages correctly, what is confirmed by the evaluated simulations.

However, these equations only hold when input messages are drawn uniformly at random. Correlation between stored messages can lead to dramatic decrease in the performance of such memories.

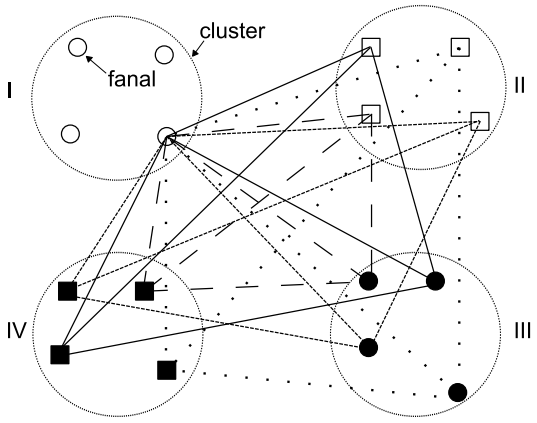

Figure 2: Network with non-uniformly distributed messages. Different shapes (filled or empty circles or squares) represent fanals belonging to different clusters, different types of lines represent connections belonging to distinct cliques. Clusters are numbered - they represent segments of messages.

3.2 Non-uniform distribution example

Figure 2 represents a case of a network made of four clusters. Fanals belonging to specific clusters are represented with different shapes. There are four fanals per cluster. The number of segments in the messages is four, each cluster corresponding to a given segment, and on each segment one of the four values is chosen. Cliques are formed by lines connecting fanals. Figure 2 depicts a situation where four messages are stored, each type of the line representing a different clique.

For a set of messages stored in the network, some fanals can have much more connections than the others. One can observe that in all of the clusters except cluster I, each node has the same number of connections. This means that on the segments II, III, IV of the messages each of the four possible values occured. However, in the cluster I, only one of the fanals is always used, i.e. the value on the first segment is constant. This simple example depicts how the distribution of data stored in the network maps to the interconnection structure.

Figure 3 shows the evolution of the error retrieval rate for a larger network with $c = 8$ and $\ell = 256$. In that specific case, half the clusters are not provided with information. This means that only four randomly chosen segments of a message are known, the remaining are erased. Hence, only four fanals are initially stimulated in four clusters. Figure 3 shows also the theoretical curve for a single iteration and the network density. Note the interest of the iterative character of the decoding process. The simulation shows that the network of $n = 2048$ fanals can store up to 15000 uniform messages of 64 bits each and retrieve them with a very high probability (error rate 0.029 for four iterations allowed) even when half the clusters are not provided with information. However, when the messages are generated from the Gaussian distribution (mean μ = 135, standard devia-

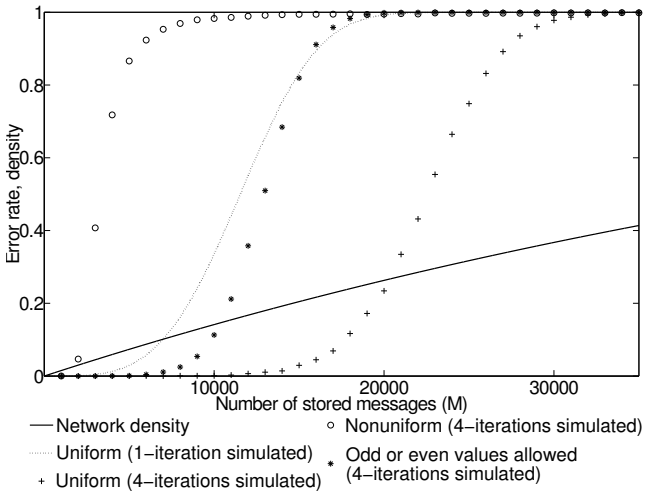

Figure 3: Evolution of the error rate as a function of the number of stored messages. The network composed of eight clusters of size 256, four randomly erased positions.

tion $\sigma = 25$), only 2000 messages can be stored (error rate 0.047). The Gaussian distribution, contrary to the uniform one, implies that on a given segment of messages some values occur much more often than the others. Figure 3 presents also a curve (indicated with asterisk) for a data with a specific correlation between the values within each message. Within each message either odd or even values are only allowed. This means that if on the first segment there is an odd value, one knows that all the other values are odd (e.g. $m = \{1\ 3\ 5\ 7\ 9\ 11\ 13\ 15\}$ in decimal). For this dataset the network can store 8000 messages and retrieve them with the same error probability as in the uniform case. One can see that the correlation in the stored data clearly shifts the curve toward lowest number of stored messages. In this example, the same network filled with normally distributed data, can store only 13% of the number of uniformly distributed messages.

As in most applications, data cannot be considered uniform, it is of first importance to introduce techniques to counterbalance the effects of correlation on performance.

4 Huffman coding and other strategies to store non-uniform data

In the following section several strategies to store non-uniform data are described. They essentially all consist in adding spatial diversity to the networks.

4.1 Adding random clusters

The first of the strategies relies on adding to the existing network, clusters filled with random values drawn from a uniform distribution. These random values stored in *random clusters* play a role of a so-called *stamp*, providing the existing clique with additional information and supporting the message retrieval process. This way the influence of local high density areas caused by non-uniform data is neutralised. Figure 4 illustrates the network from Figure 2 with

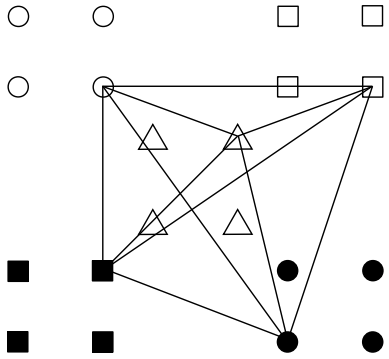

Figure 4: Network with one random cluster (represented by triangles) added.

one random cluster added and one message stored. When using this technique, each message comes with an additional segment (or several segments) which holds a value randomly generated from an arbitrary chosen range. Then, during the retrieval, only a subset of known non-random segments is used to initially stimulate the network. Through the established connections they stimulate all the other clusters, random clusters as well. Then, the latter stimulate the others and give additional support to the retrieval process.

The main advantage of this technique is the fact that no additional processing on the stored data is needed, the values that are stored in the network are directly used for the initial stimulation. Thanks to the additional clusters, the necessary treatment is done by the network itself.

Similar technique is introduced in (Knoblauch, Palm, and Sommer 2010). The authors propose adding to a feedforward neural network an additional intermediary layer filled with random patterns, in order to improve the performance of a single-layer model in case of non-random data. Since the two models have much in common, this technique is treated as a reference for the rest of the introduced strategies.

After explaining other techniques, the following section presents how they improve the performance compared to adding random clusters.

4.2 Adding random bits

Another category of the proposed strategies relies on adding random uniformly generated bits to the existing data rather than whole random values stored in separate clusters. As a consequence, the structure of the network remains the same, only the size of clusters is modified since the range of values is expanded by a number of random bits. In other words, single fanal is no more associated with a given value. For instance, adding one random bit to a segment implies randomly choosing between two fanals associated with a given value.

Besides adding bits simply drawn from the uniform random distribution another approach to generate additional bits is proposed. To each value always the least used combination of bits (or one of the least used) is added. The rule is illustrated with Figure 5. For each of the values coded on

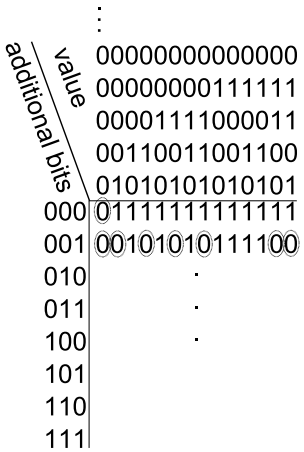

Figure 5: Adding least used combination of bits. The additional bits that are marked can be added to the initial values.

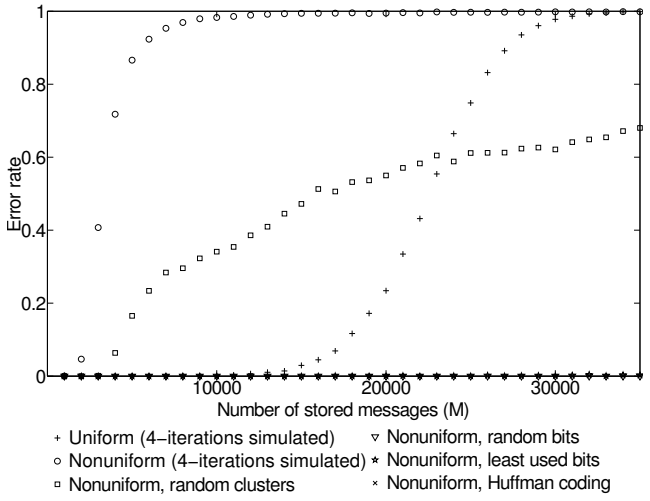

Figure 6: Evolution of the error rate when using proposed strategies as a function of the number of stored messages compared to the uniform and non-uniform cases. Four positions are randomly erased. The material used for each strategy is comparable.

8 bits a table is created where the number of occurrences of each additional bits combination is stored. For instance, for the first value consisting of only zeros either 000 or 001 can be chosen. However, for value 00000001 only 001 can be added since 000 is already used once. This procedure continues for all the positions in all the messages.

As it is shown in the following section, this technique offers much better performance compared to random clusters strategy when using comparable material. However, it requires changing the data by adding random bits before storing it in the network. This does not imply higher computational complexity compared to adding random clusters, since both methods require only random numbers generation. Nevertheless, manipulating the data before storing it, may be constraining in some applications.

4.3 Using compression codes

The last proposed strategy is to apply algebraic compression codes. In this work Huffman lossless compression coding is proposed. This technique allows to minimize the average number of coding symbols per message (Huffman 1952). Note that for some datasets arithmetic coding may perform better than Huffman code, see (Bookstein and Klein 1993) for comparison.

Huffman coding produces variable length codewords - the values that occur most frequently are coded on a small number of bits, whereas less frequent values occupy more space. One dictionary is constructed for each segment of all the messages. Such procedure results in variable length segments, the most often occuring values being the shortest. Therefore, the sizes of the frequent values that break the uniformity are minimized. The free space obtained within each segment is filled with random uniformly generated bits. Now, the most often appearing values are associated with the largest number of randomly chosen fanals. Decoding is possible thanks to the prefix-free property, that is a set of bits representing a symbol is never a prefix of another codeword. Strictly speaking, in order to decode the retrieved encoded message, each segment is compared bit-by-bit with its dictionary. When a codeword is met, one knows that these bits are useful, the remaining being the random ones.

The simulations prove that using compression codes is the most effective from the presented techniques. Thanks to the properties of the used coding technique, the frequent values are effectively redistributed across a group of fanals. Consequently, the influence of local high density areas is minimized. Compression codes, however, require additional operations related to coding and decoding. In some applications the fact that in order to compute the Huffman code the entire data set is required can also be a limitation. In this case a variation of Huffman coding such as Adaptive Huffman coding (Vitter 1987) can be used. In this technique, the code is built as the data is transmitted, with no initial knowledge of the distribution.

5 Performance comparison

In this section performance of all the introduced strategies is evaluated. The simulations are performed for the same network as in section 3 and the same non-uniform distribution.

Figure 6 shows the improvement in performance when using the proposed techniques. To prove that adding a certain material improves the network functioning, seven random clusters of 5000 fanals each are added. Compared to non-uniform case without using any technique the improvement is clear. The function is non-monotonic since the significant part of the network is filled with random values. To compare the used techniques, each of them is classified by the amount of material it uses. The used material is considered to be the number of possible connections. In case of an implementation, be it software or hardware, it implies the number of information (bits) that is needed to indicate whether the connections between nodes are established or not. The rest of the techniques (random bits, least used bits, Huff-

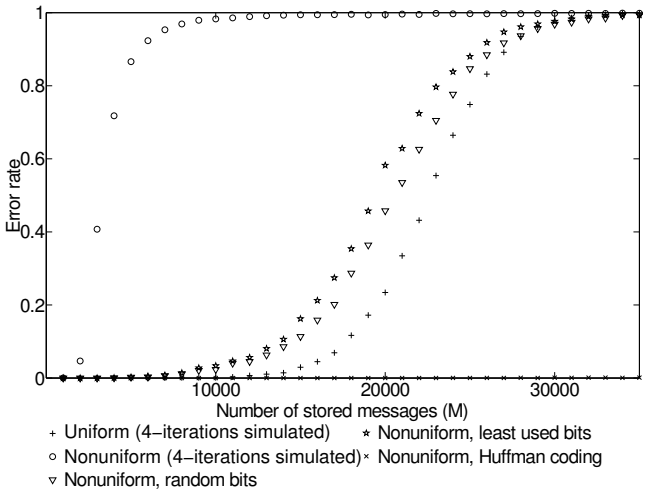

Figure 7: Evolution of the error rate when using proposed strategies as a function of the number of stored messages compared to the uniform and non-uniform cases. Four positions are randomly erased. The minimal material to obtain close-to-uniform performance is used.

Table 1: The limiting number of messages that can be stored until the error rate reaches 0.1. Two additional bits used.

Technique	No. messages
Least used bits	14000
Random bits	15000
Huffman coding	117000
Uniform messages	220000

man coding) use the closest additional material compared to the random clusters strategy. This is obtained for eight clusters of 4096 fanals which corresponds to four additional bits. These strategies offer much better performance using comparable material, obtained error rate always remains near to zero.

Figure 7 presents results when using minimum material to approach the performance close to the uniform case. When the random bits and the least used combination strategies are applied, two additional bits are enough to get close to the uniform data case. This means that the size ℓ of the clusters equals 1024 and the material used accounts for 4.9% of the material used for random clusters strategy. When three bits are added (not shown on the plot) the performance gets already much better than for the uniform messages, therefore two bits are the minimal necessary material needed. The curve for Huffman coding technique also uses two additional bits (two bits turned out to be the minimal material for the random bits, least used bits and Huffman coding techniques). In this case the gain in performance is significant, the error rate always stays close to zero. For the used data distribution the length of the messages sometimes exceeds 72 bits (8 clusters $\times$ (8 bits + 1 random bit)) and consequently, adding less than two bits is not possible. However, for different data sets minimizing the used material could still be feasible.

Figure 7 does not show the limiting number of messages for Huffman coding technique. Therefore, additional simulations for larger numbers of stored messages were carried out. Table 1 shows a comparison of the strategies when two additional bits were used (corresponds to Figure 7) and the error rate reaches 0.1. It also compares the results to the case when uniform messages are stored in the network of the same size ($c = 8$, $\ell = 1024$).

6 Conclusion

In this work the performance of the network (Gripon and Berrou 2011b) in case of non-uniformly distributed messages is evaluated. The analyses showed clearly the importance of adapting the network to such a kind of data. Several strategies to avoid performance degradation are proposed and evaluated. One of them relies on supporting the cliques with additional signals coming from the added clusters. The principle of the other techniques is to spread the same values across a group of fanals. In order to apply these recently introduced memories in practical applications, adapting the network to non-uniform messages is indispensable, as real-world data is not necessarily uniformly distributed.

Compared to the technique used for the similar network model, other proposed strategies offer performance improvements. Especially the strategy that uses Huffman coding offers great performance enhancements, since it minimizes the length of each message segment based on its distribution. Consequently, frequent values are coded on a few bits which allows adding random bits determining which fanal from a group of available is used. This way, often appearing values are efficiently redistributed across larger groups of fanals, limiting the negative influence of high density areas. However, Huffman coding requires constructing the dictionaries in order to code and decode messages.

The application determines which technique should be used. If one can afford transforming data before storing it in the network either random bits or compression codes should be used. Otherwise, one should use random clusters technique where only initial data is used when stimulating the network. Alternatively, when the used material is constraining, random bits and compression codes are more efficient. Nevertheless, Huffman coding implies additional cost of coding/decoding which means that in some cases random bits offer better performance/cost trade off.

Future work may include exploring some self-adapting techniques built in the network architecture. For instance, one can envision limiting the density on fanal level and when a specified value is reached using a new fanal instead. Another possibility to be considered is adapting the size of cliques with respect to data distribution, giving more support to the cliques containing frequent values.

7 Acknowledgments

This work was supported in part by the European Research Council (ERC-AdG2011 290901 NEUCOD).

References

Ackley, D. H.; Hinton, G. E.; and Stejnowski, T. J. 1985. A learning algorithm for Boltzmann machines. *Cognit. Sci.* 9(1):147–169.

Agrawal, B., and Sherwood, T. 2006. Modeling TCAM power for next generation network devices. In *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 120–129.

Aliabadi, B. K.; Berrou, C.; Gripon, V.; and Jiang, X. 2013. Storing sparse messages in networks of neural cliques. *IEEE Transactions on Neural Networks and Learning Systems* PP(99):1 – 10.

Bookstein, A., and Klein, S. 1993. Is Huffman coding dead? *Computing* 50:279–296.

Gripon, V., and Berrou, C. 2011a. 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*, 54–58.

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

Gripon, V., and Rabbat, M. 2013. Maximum likelihood associative memories. In *Proceedings of Information Theory Workshop*, 1–5.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci* 79(8):2554–2558.

Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers* 40(9):1098–1101.

Jarollahi, H.; Gripon, V.; Onizawa, N.; and Gross, W. J. 2013. A low-power content-addressable-memory based on clustered-sparse-networks. In *Proceedings of 24th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2013)*.

Jouppi, N. P. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th annual international symposium on Computer Architecture*, ISCA '90, 364–373.

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

Larras, B.; Boguslawski, B.; Lahuec, C.; Arzel, M.; Seguin, F.; and Heitzmann, F. 2013a. Analog encoded neural network for power management in MPSoC. In *Proceedings of the 11th International IEEE New Circuits and Systems Conference*, NEWCAS '13, 1–4.

Larras, B.; Lahuec, C.; Arzel, M.; and Seguin, F. 2013b. Analog implementation of encoded neural networks. In *ISCAS 2013 : IEEE International Symposium on Circuits and Systems*, 1 – 4.

Lin, L.; Osan, R.; and Tsien, J. Z. 2006. Organizing principles of real-time memory encoding: Neural clique assemblies and universal neural codes. *Trends Neurosci.* 29(1):48–57.

Lin, C. S.; Smith, D. C. P.; and Smith, J. M. 1976. The design of a rotating associative memory for relational database applications. *ACM Trans. Database Syst.* 1(1):53–65.

Lou, N. H. W. C. C., and Chen, J. 2001. Design of multi-field IPv6 packet classifiers using ternary cams. In *Proc. IEEE GLOBECOM*, volume 3, 1877–1881.

Pagiamtzis, K., and Sheikholeslami, A. 2006. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits* 41(3):712–727.

Palm, G. 2013. Neural associative memories and sparse coding. *Neural Netw.* 37:165–171.

Papadogiannakis, A.; Polychronakis, M.; and Markatos, E. P. 2010. Improving the accuracy of network intrusion detection systems under load using selective packet discarding. In *Proceedings of the Third European Workshop on System Security*, EUROSEC '10, 15–21.

Salavati, A. H., and Karbasi, A. 2012. Multi-level error-resilient neural networks. In *ISIT*, 1064–1068. IEEE.

Vitter, J. S. 1987. Design and analysis of dynamic Huffman codes. *J. ACM* 34(4):825–845.

Willshaw, D. J.; Buneman, O. P.; and Higgins, L. H. C. 1969. Non-holographic associative memory. *Nature* 222:960–962.