Weightless Neural Models: A Review of Current and Past Works

Teresa B. Ludermir †, André de Carvalho ‡, Antonio P. Braga §, and Marcílio C. P. de Souto ¶
† Depto. de Informática, Univ. Federal de Pernambuco, Recife, Brazil
‡ Laboratório de Inteligência Computacional, Univ. de São Paulo, São Carlos, Brazil
§ Depto. de Engenharia Eletrônica, Univ. Federal de Minas Gerais, Belo Horizonte, Brazil
¶ Dept. of Electrical Engineering, Imperial College, London, UK

Abstract

This paper presents a survey of a class of neural models known as Weightless Neural Networks (WNNs). As the name suggests, these models do not use weighted connections between nodes. Instead, a different kind of neuron model, usually based on RAM memory devices, is used. In the literature, the terms "RAM-based" and "n-tuple based" systems are also commonly used to refer to WNNs. WNNs are being widely investigated, motivating relevant applications and two international workshops in the last few years.

The paper describes the most important works in WNNs found in the literature, pointing out the challenges and future directions in the area. A comparative study between weightless and weighted models is also presented.

1 Introduction

The neural computing models described in this paper are based on artificial neurons which have binary inputs and outputs and no weight between nodes. Neuron functions are stored into look-up tables, which can be implemented using commercially available Random Access Memories (RAMs). Instead of adjusting weights, learning on Weightless Neural Networks (WNNs) generally consists of changing the contents of look-up table entries, which results in highly flexible and fast learning algorithms. Both supervised [23, 47, 52, 58, 63] and unsupervised [15, 38, 42, 85] learning techniques have been used with WNNs. The effectiveness of WNN models for real-world applications has been demonstrated in a number of experimental studies [37, 56, 62, 74, 87, 90, 104].

The high speed of the learning process in WNNs is due to the existence of mutual independence between nodes when entries are changed, which contrasts with conventional weighted-sum-and-threshold neurons. In weighted models, training is more complex, since changing weights to train a given input-output mapping changes the node's behaviour to other patterns learned previously. The flexibility comes from the high degree of functionality of individual weightless neurons. Flexibility, easy parallel implementation, high speed of learning and the possibility to implement real networks using commercially available RAMs are considered to be the main advantages of WNNs over their weighted counterparts.

There is some misunderstanding in the literature about the definitions of Boolean Neural Networks (BNN) and WNNs. BNN are those networks that handle only Boolean values at their inputs and outputs. They may have nodes based on the model proposed by McCulloch and Pitts [75], with weights associated to their connections. WNNs, as the name suggests, do not have adaptive weights associated to their connections. They can also be seen as having fixed binary weights. Thus, although WNNs are also BNN, the opposite is

[0]This work is supported by CAPES, CNPq, FACEPE, and FAPESP. Updates, corrections, and comments should be sent to Teresa B. Ludermir at tbl@di.ufpe.br.

not true. As an example, Hopfield networks [57] are BNNs, but not WNNs. WNNs have their origin on the N-tuple sampling machines described by Bledsoe and Browning in the late 50s [27]. Aleksander took a great deal of interest in adaptive learning networks using N-tuple sampling machines; he suggested a universal logic circuit as the node of a learning network [4]. He also introduced the RAM node and the SLAM (Stored Logic Adaptive Microcircuit), which were designed for research purposes before the availability of integrated circuit memories.

By the late 70s, the construction of a large network able to deal with high quality images became feasible as RAM elements were much cheaper and had higher capacity. The construction of the WISARD (Wilkie, Stonham & Aleksander's Recognition Device) was then possible, with its first prototype completed in 1981 [14]. The machine was patented and produced commercially from 1984. After WISARD, other weightless models were proposed in the literature, such as PLNs, pRAMs, GSNs, and GRAMs [8, 18, 22]. Also, in the literature, the terms "RAM-based" and "n-tuple based" have been used to refer to WNNs. These models are described in more detail in the next sections.

2 The RAM node

In contrast to biologically motivated nodes, RAM nodes were initially designed as engineering tools to solve pattern recognition problems. An N input RAM node (Figure 1) has 2^N memory locations, addressed by the N-bit vector $\mathbf{a} = \{a_1, a_2, ..., a_N\}$. A binary signal $\mathbf{I} = \{I_1, I_2, ..., I_N\}$ on the input lines will access only one of these locations, that is, the one for which $\mathbf{a} = \mathbf{I}$. The bit, $C[\mathbf{I}]$, stored at this activated memory represents the output of the node, that is, $r = C[\mathbf{I}]$. In other words, the Boolean function performed by the neuron is determined by the contents of the RAM. There are 2^{2^N} different functions which can be performed on N address lines and these correspond to the 2^N states that the RAM node can be in. Thus, a single RAM can compute any Boolean function of its inputs.

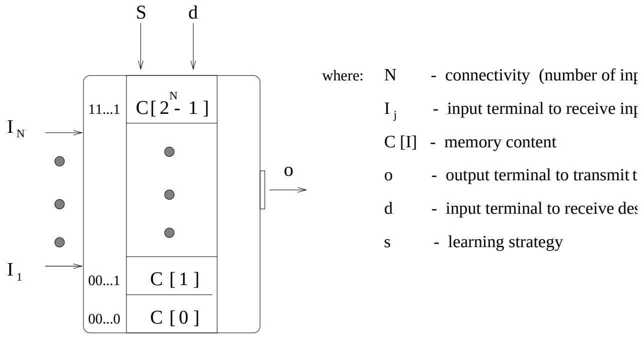

Figure 1: Typical RAM node

Learning in a RAM node takes place simply by writing into the corresponding look-up table entries. This learning process is much simpler than the adjustment of weights. The RAM node, as defined above, can compute all binary functions of its input while the weighted-sum-and-threshold nodes can only compute linearly separable functions. There is no generalisation in the RAM node itself. Generalisation may be understood as the ability to provide the correct output for patterns not seen in the training phase. Conventional RAM nodes produce the correct output only for those input patterns stored in the training phase. However, there is generalisation in networks composed of RAM nodes [5]. Generalisation can be introduced by considering the hamming distance from training patterns or masks.

Although RAM nodes deal only with binary values, by preprocessing the input data, these nodes can be extended to handle scalar values. The thermometer coding [10], the rank coding [16], the interval coding or Minchinton cells [26], and more recently the CMAC coding [69], are examples of methods which can be used to process scalar data for RAM networks. The method in [69] has been shown suitable for different applications [90].

In a typical RAM network, that is, a single-layer architecture, RAMs are taught to respond with a 1 for those patterns in the training set, and only for those patterns. An unseen pattern is classified in the same class of the training set if all RAMs output 1s. This simple architecture divides the set of all possible patterns into those that are in the generalisation set, and those that are not. If more than two categories are required, many RAM networks are used together, each net trained to respond to one class of pattern. These networks are modified so that instead of having a logic gate to combine the RAMs' outputs, the decision is left to the *maximum response detector*. These modified RAM networks, which are used both in SLAMs and WISARD [14], are called *discriminators* (Figure 2)

2.1 WISARD

The advances in integrated circuit technology during the 70s led Aleksander and Stonham to implement N-tuple machines using conventional RAM devices. This approach led to the description and construction of WISARD [14], a general purpose pattern recognition machine. Recently, a comparison between WISARD and other classification methods (including neural networks) was performed for various classification problems [90, 91]. The results obtained, together with previous ones [13, 88, 98], show that the underlying principle of RAM networks (WISARD) is a powerful method. This method was also analysed into a probabilistic framework [31, 72, 92].

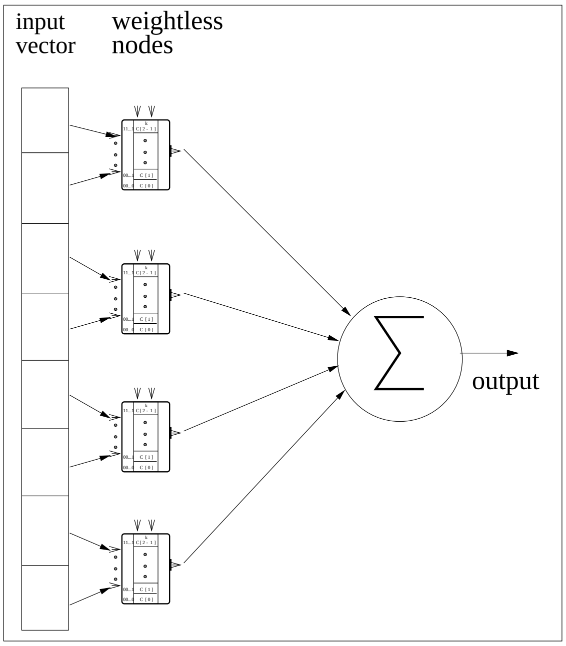

Figure 2: Discriminator

The basic element of WISARD is a bit-addressable RAM discriminator that is formed by a layer of k N-input RAMs, each one storing 2^N one-bit words (Figure 2). Each of these k RAMs is addressable by an N-tuple, which is selected randomly from an array of binary values forming the input to the network. The way that the network is connected to its input is called the input mapping, or the connectivity pattern. Although the input mapping is chosen at random, such a mapping is often a fixed parameter of the network [13, 90]. At the discriminator's output, an adder sums the outputs of the RAMs, producing what is called the discriminator's response r. Before training takes place, all the $k2^N$ one-bit words are set to zero, and training consists in their modification.

Training a kN input pattern $\mathbf{v}$ is carried on by presenting $\mathbf{v}$ at the discriminator's inputs and by setting to 1 all the words accessed by $\mathbf{v}$ in all k nodes. If $\mathbf{v}$ is presented to the discriminator at a later time, such previously written locations will all be accessed and every node will respond with 1, leading the discriminator response to be maximum ($r = k$). If a noisy version $\mathbf{u}$ of $\mathbf{v}$ is presented, r is a function of the number of previously written locations that are accessed by $\mathbf{u}$, which is proportional to the similarity between $\mathbf{v}$ and $\mathbf{u}$. Therefore, although individual RAMs do not generalise, a WISARD discriminator does, since it is able to classify unknown patterns, given the knowledge acquired by the learning of the training set.

A WISARD system is built by grouping together a set of discriminators, each one being responsible for recognising a different class of patterns. During training of a pattern $\mathbf{v}$, writing occurs only in the RAMs of the corresponding discriminator D_i. After training is carried out individually for all the classes, the recognition of an unknown test pattern $\mathbf{u}$ is made by presenting $\mathbf{u}$ concurrently to all the discriminators' inputs and by checking each response r. The pattern is assigned the class corresponding to the discriminator with the highest response r.

The performance of the N-tuple method depends mainly on the size of the N-tuple chosen [13, 89, 100, 101]. Experimental results, for instance, showed that a bigger size of N is better, until it approaches an impractical large size, though a value of $N = 8$ turned out to be enough for many applications [89, 90]. However, in general, the value of N is constrained by many considerations, where some of them might not be simultaneously satisfied in all situations. In such cases, a simple adjusting in N alone will not significantly improve the network performance [90].

Another important point is the training process. For example, the method of training described above ensures that all the node functions output one when presented with the specific N-tuple in the training pattern. Thus, problems arise when most memory locations are set to 1 so that a pattern also yields $r = k$ in other discriminators. This is known as the saturation problem [100]. So, in order to improve the performance of the N-tuple method, others learning algorithms have been proposed. These methods were based initially on the idea of maximum likelihood [28, 24, 97, 100].

More recently, in order to overcome the saturation problem, Tarling and Rohwer [98] proposed a simple modification in the original N-tuple training algorithm. Their experiments showed that the modification brought a decrease in the level of saturation and increase in generalisation accuracy over an identical experiment performed with the original algorithm. A further training algorithm, called Stochastic Minimisation Algorithm (SMA), was proposed in [31]. The SMA was used for estimation of the Vapnik-Chervonenkis (VC) dimension [102] of the N-tuple classifier, but can also be applied to real-world data.

A different kind of learning algorithm, involving changes in the connectivity pattern, was introduced by Jorgensen *et al.* in [63]. They proposed a measure, which combines the concept of cross-validation and Shannon information theory, to be used during training to select the best connections needed to achieve a given performance. An example of the application of this technique can be found in [62]. A similar problem, but more concerned with storage cost, was approached in [78]. Such a study described several ways in which memory requirements of N-tuple systems can be reduced. Finally, it is important to point out that, although these modified methods described above, could achieve a better performance, they often reduce the speed and simplicity advantages of the original algorithm.

The use of RAM-based networks incorporating N-tuple methods in associative memories was also investigated in [23]. This work was motivated by the generalisation problems faced by Willshaw nets [106]. Such problems were reduced by the development of an associative architecture which combines an N-tuple front-end processor with two-stages of Willshaw net. This architecture is called ADAM (Advanced Distributed Associative Memory) [23]. ADAM uses an N-tuple technique as a pre-processor (a RAM network) to cope with non-linearities and to reduce the saturation of the matrix memories. Such a model has been shown to have a large storage capacity [17] and to be tolerant to binary errors, particularly when small sized tuple processors are used [29].

An important aspect about RAM networks is that although there is no ambiguous information concerning RAM locations which contain 1, the same does not hold for those that are left in the 0 state. An unknown vector accessing an entry that has 0 could mean either (1) that such vector is a counter-example of the corresponding class; or (2) that no information was provided during training for that class feature corresponding to the entry accessed. It is not easy to define which one is true. Another discriminator holding a value 1 at the same location indicates a counter-example. In the second case, the vector could be an example of the class, but since no information was given, the RAM will output 0 instead of 1. To overcome this problem, Kan and Aleksander [64] developed the word-addressable logic node called Probabilistic Logic Node (PLN), which will be described in the next section.

3 PLN

A PLN node differs from a RAM node in that a 2-bit number (rather than a single bit) is now stored at the addressed memory location. The content of this location is turned into the probability of firing (i.e., generating 1) at the overall output of the node. In other words, a PLN consists of a RAM node augmented with a probabilistic output generator. Thus, like in a RAM node, the N binary inputs to a PLN node form an address to one of the 2^N addressable locations. Simple RAM nodes then output the stored value directly.

In contrast, in a PLN, the content at this address is passed through the output function which converts it into a binary node output. Such a content could be either 0's, 1's, or u's. The undefined state u implies on the node flipping its output between 0 and 1 with equal probability. The use of a third logic value, u (undefined), makes possible the use of an "unknown" state in the operation of WNNs architectures. This value is stored in all the memory contents before the learning phase, indicating the ignorance of the network before it was trained. The output of the PLN node is given by:

$$r = \begin{cases} 0 & \text{if } C[\mathbf{I}] = 0 \\ 1 & \text{if } C[\mathbf{I}] = 1 \\ \mathsf{random}(0,1) & \text{if } C[\mathbf{I}] = u \end{cases} \tag{1}$$

where $C[\mathbf{I}]$ is the content in the address position associated with the input pattern $\mathbf{I}$ and random(0,1) is a random function that generates zeros and ones with the same probability.

Before the PLN, all the RAM-based neural models which learnt by updating the values stored in the memory contents were single-layer architectures. Aleksander [6] proposed the use of PLN nodes in multi-layer architectures, called pyramids, which can be seen in Figure 3. A pyramid is a tree structure in which each layer usually has a fixed number of neurons with a fan-out of 1 and low fan-in. Pyramids have lower functionality than a single neuron covering the same input pixels. On the other hand, if compared to this single neuron, a pyramid reduces the size of memory needed and increases generalisation [64].

Training PLN networks becomes a process of replacing u's with 0's and 1's so that the network consistently produces the correct output pattern in response to training pattern inputs. At the beginning of training, all stored values in all nodes are set to u, and thus the net's behaviour is completely unbiased. A generic gradient descendent learning algorithm, proposed by Myers, [83], uses several presentations of the training set to teach pyramids of PLNs by using a reward and a punish phase. In [3], Al-Alawi & Stonham introduced a learning procedure, the back-propagation search algorithm, for multi-layer WNNs and applied it to pyramids of PLNs. Such an algorithm is able to store all patterns in the training set without disruption, if at least one of the functions that satisfy the data set is performed by the pyramid. They also studied the functionality of pyramidal structures. Such a functionality was shown to be greater than that of weighted-sum-and-threshold neurons in [80]. Zhang *et al.* [108] presented both a training algorithm that can be used with pyramids of PLNs and a formula for computing the average number of steps that the algorithm converges (when there is a solution).

A more general approach for the computational complexity of training pyramids was presented in [45], where this problem was shown to be an NP-complete problem. Nevertheless, it was also shown that a constrained class of pyramids, called multi-pyramidal architectures (a set of disjoint tress with bounded depth), can be trained in polynomial time. Studies on the storage capacity of pyramidal architectures were presented in [1, 86]. In [86], such a problem was analysed into the the context of information theory. The

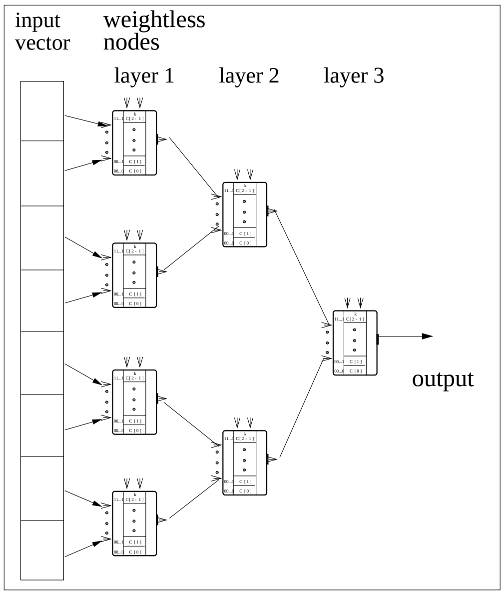

Figure 3: PLN networks

analysis in [1] showed that, in pyramids, the reduction in storage capacity is bigger than the reduction in cost (the total amount of sites in a network) when the node fan-in is diminished.

The use of three-logic values and pyramid structures by PLN networks represented a breakthrough in the WNN research. Other WNNs models were soon created incorporating these ideas, such as the MPLN and pRAM. These nodes, which are similar in many ways, were simultaneously and independently developed in [82, 83] and [49, 51], respectively.

4 MPLN AND pRAM

The development of the PLN led to the definition of the m-state PLN or MPLN [82, 83]. The main difference between the MPLN and the PLN is that the first allows a wider, but still discrete, range of probabilities to be stored at each memory content. Also, the output function for an MPLN could be, in addition to the probabilistic function, a linear function, or the sigmoid function [83]. One result of extending the PLN to MPLN is that the node locations may now store output probabilities which are more finely graded than in the PLN.

An MPLN, for instance, could output 1 with 15% probability under a certain input. Based on this, learning can allow incremental changes in stored values. This way, one reset does not erase much information — erroneous information is discarded only after a series of errors. Similarly, new information is only acquired after a series of experiences indicates its validity. So, the reinforcement learning procedure for PLNs was extended to take this into account and was applied to model delay learning in invertebrates [81].

In [99], Taylor proposed a model of noisy neurons that presents equations which incorporate and formalise many known properties of living neurons. The evolution of such a model was shown to be equivalent to that of networks of noisy (*probabilistic*) RAMs or pRAMs [49]. The pRAM is also an extension of the PLN like the MPLN, but in which continuous probabilities can be stored, that is, value in the range $[0, 1]$. This kind of node outputs 1 (spike) with frequency related to the memory content being addressed. The pRAM was further extended, so as to be able to map continuous input to binary outputs. Such an extension is called the *integrating* pRAM (i-pRAM) [51]. Another Boolean unit with the capability of dealing with continuous inputs, called cubic node, was also proposed in [55].

An N input pRAM node A (Figure 4), as in the definition of the RAM node in Section 2, has 2^N

memory locations addressed by a vector **a**, where a binary signal **I** in the input lines will access only one of these locations. But, unlike the RAM node, the value $C[\mathbf{I}]$ stored at this activated location represents the probability that a spike value 1 will be produced on the output line r, given **I** as input. Hence, in this binary input case, the mean value y of the pRAM output is directly given by the memory content $C[\mathbf{I}]$ being accessed, that is, $y = C[\mathbf{I}]$.

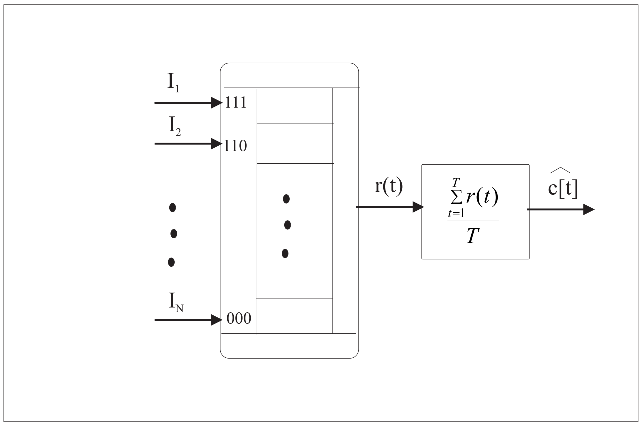

Figure 4: The process of output computation for a binary-input pRAM (taken from [48])

Now, let $\mathbf{x} = \{x_1, x_2, ..., x_N\}$ ($\mathbf{x} \in [0,1]^N$) be a real-valued input for A. In this case, each x_i ($i = 1, ..., N$) could be represented by a stream of pulses I_j, in which x_j defines the probability that 1 will occur in the jth position of **I** ($P(I_j = 1) = x_j$). So, unlike in the original binary input pRAM or in any other weightless node, in the real-valued pRAM (Figure 5) there is no direct access to only one specific look-up table entry to identify which probability of firing is associated with the current input. But, over $t = 1...T$ time steps a spread of locations will be accessed and contribute stochastically with a spike value $r = 1$ or $r = 0$ to the output stream. In other words, over time the node will access a number of memory locations and, over time, fire with a mean rate depending on those nodes values. In such a case, the mean value of the output is given by:

$$\begin{aligned} y &= \textstyle\sum_{\mathbf{a}} C[\mathbf{a}] \prod_{j=1}^{N} [a_j x_j + (1 - a_j)(1 - x_j)] \\ &\equiv \textstyle\sum_{\mathbf{a}} C[\mathbf{a}] X_{\mathbf{a}}(\mathbf{x}) \end{aligned} \tag{2}$$

where $C[\mathbf{a}]$ is the memory content of site **a**, and $X_{\mathbf{a}}$ is the probability that address **a** is accessed. Such a probability is computed based on probability distribution functions defined by **x**. This output function can be seen as a polynomial containing products up to N input signals. Thus, in the PDP terminology [94], these pRAM nodes are sigma-pi units. Networks with pRAM nodes can also have the ability to approximate universal function [48].

When hardware is used, the output y of the pRAM is approximated by the time-average (over some period T) of successive output bits (output stream), independent of whether the input is continuous or not:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} r(t) \tag{3}$$

If $T = 1$ and $X_{\mathbf{a}}(\mathbf{x}) \in \{0,1\}$, the pRAM is equivalent to the MPLN. With $T > 1$ and $X_{\mathbf{a}}(\mathbf{x}) \in [0,1]$, the pRAM model is much more complex, and this is the main difference with the MPLN: the pRAM can output frequencies truly dependent on the probabilities stored in the memory contents. Also, in this continuous version the pRAM exhibits generalisation properties.

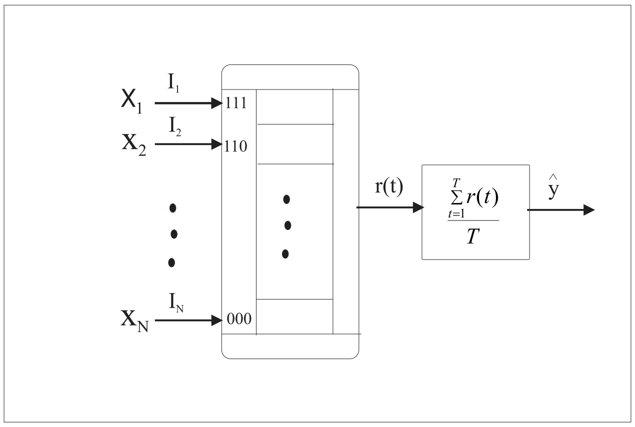

Figure 5: The process of output computation for a pRAM with real-valued components (taken from [48])

In general, pRAMs have been used in pyramids [54, 87] and trained by means of reinforcement procedures [50, 51, 53]. The convergence of the reinforcement procedure for pRAM networks was investigated in [52], and the generalisation properties of these networks were addressed in [40]. An important aspect about pRAMs and their reinforcement training algorithm is the fact that they are hardware implementable, and chips are commercially available [41]. More recently, Gorse *et al.* [48] proposed a spike-based reinforcement learning algorithm which allows continuous functions to be learned by neural networks and applied it to pRAMs networks.

A different way to implement a probabilistically based weightless node is not to use continuous firing rates, but to pass the probability of the node firing using a Boolean value. This method, used in the GSN, will be described in the following section.

5 GSN

The Goal Seeking Neuron (GSN) was designed to maximise the storage efficiency of values in its memory, by storing new data without disrupting previously learned information. Unlike a PLN, a GSN can input, store and generate values equal to 0, 1 or u. If there is at least one value u on the input terminals of a GSN, then a *set* of memory contents will be addressed. This set of addresses is derived from the original address by changing the u values to 0 and 1. For instance, suppose that a four-input GSN receives the vector $\mathbf{I} = \{0, 0, u, 1\}$ as input. Since there is a u value in the third position of $\mathbf{I}$, two addresses in the GSN will be accessed: $\mathbf{a}_1 = \{0, 0, \mathbf{0}, 1\}$ and $\mathbf{a}_2 = \{0, 0, \mathbf{1}, 1\}$. Also, unlike the output function of the PLN that outputs either 0 or 1 (even when a u location is addressed), in a GSN such a function outputs 0, 1, or u according to the number of these values stored in the addressed memory locations.

GSN neurons have been used in three different architectures, each one suitable for a different range of applications. These classes of architectures are GSN feedforward (GSN^f), GSN feedback (GSN^{fb}), and GSN self-organising (GSN^s) [47]. Such architectures are usually formed by a fixed number of pyramids, known as multi-pyramidal networks (Figure 6), where each pyramid covers a subset in the pixels of the input field. The use of genetic algorithms to optimise the topology of GSN-based multi-pyramidal networks was proposed by Martins in [73]. These multi-pyramidal GSN networks often have three operation phases, each associated

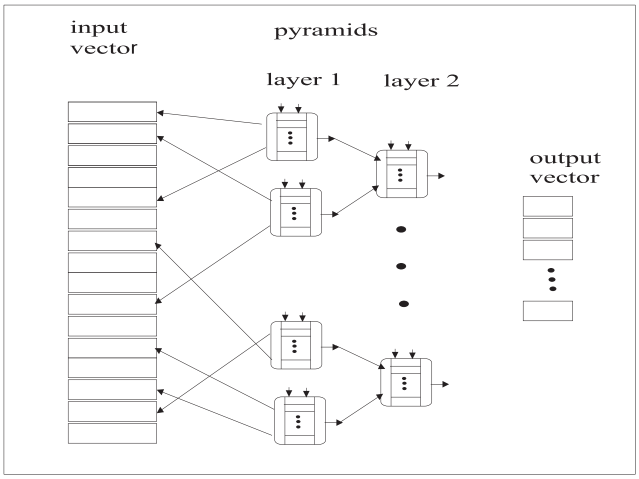

Figure 6: Multi-pyramidal architecture

with a specific goal: a validating phase, a learning phase and a recall phase. Before the network training, all of its memory contents in the network are filled with u values. The validating phase, which implements a kind of depth-first search with backtracking, could be seen as a sub-phase of the learning phase. The purpose of the validating phase is, for each input in the training set, to look for a path through the pyramids that results in a u as output when the pyramids are fed with the input pattern. In this case, the u value at the output terminal of a pyramid means that any desired response for such a pyramid could be learned, that is, 0 or 1. On the other hand, when this validating value is either 0 or 1, the pyramid can be only taught if its respective desired response is equal to the validating one. Otherwise, there would be a disruption of information previously stored. So, based on this, the learning phase for a given pattern is only activated if no contradiction has happened in the validating phase.

In the learning phase, the contents of the locations which led the pyramids to produce the desired response are modified to allow the network to learn the current association. Different algorithms have been proposed to teach GSN neural networks [30, 43]. The basic characteristic of these algorithms is that the training set is often presented in a single epoch, that is, one-shot learning algorithms. During the recall phase, when the network is presented with an unseen pattern, the output of a node will be the defined value (i.e., 0 or 1) which has the highest occurrence in the addressed memory locations. This kind of output function is due to the fact that in a GSN, for a given input, more than one site might be accessed at the same time. In the case when both 0 and 1 have occurred with the same frequency, the output of the node will be the undefined value u.

GSN-based architectures form the basis for recently developed architectures such as the Self Organising Feature exTractor (SOFT) [42] and the Boolean Convergence Network (BCN) [58]. SOFT was designed to be used as a feature extractor for GSN-based neural classifiers [44]. BCNs were proposed to overcome problems such as the lack of cross connection faced by the previous GSN neural architectures (pyramids). BCN architectures were further extended and led to the development of the Generalised Convergent Network (GCN) [59], which can be used with any kind of weightless node. Finally, based on both BCNs and GCNs, the Probabilistic Convergent Network (PCN) was proposed [60].

The following section presents the GRAM model, which aims to increase the generalisation of WNNs at the node level.

6 GRAM

Since only the locations accessed by the trained patterns are modified during learning, a PLN suffers from hesitation when the training set is small. This is because too many entries are left in the u-state. Another characteristic of PLNs and RAM-based neurons mentioned earlier, with exception of the pRAM, is that they only generalise at the network level: individual nodes do not have generalisation properties. To improve WNNs models, Aleksander [7] suggested a modification in the PLN learning algorithm by including a *spreading phase* after patterns are stored.

The effect of *spreading* is to create classes or clusters having the stored patterns as their centroids. The spreading algorithm is carried out by searching the space in a radial basis having each stored pattern as a reference. The process stops at the maximum number of iterations defined by the user. In the first step of the algorithm, only those u-locations that are at Hamming distance 1 to each one of the stored patterns are assigned the nearest class or left in the u-state if there is contradiction. In the second step, only those locations that are at Hamming distance 2 are affected, and so on. The expansion by spreading of each one of the classes defined by the training set patterns is limited by the expansion of the others. Crosstalk between two classes happens when their boundaries meet each other due to successive spreading phases. In [34], Braga and Aleksander developed a model to predict the percentage of the space that is in the overlap between two classes in N-dimensional Boolean space as well as the probability of each one of the two classes occurring.

In other words, the spreading algorithm generalises information from the presented training set, since it attributes the class of the nearest stored pattern to those locations for which no information was given. A novel pattern that addresses a spread region produces the output of the corresponding spread class. If a contradictory region is accessed the output is randomly selected between 0 or 1. Since individual nodes with such learning scheme do generalise (implicitly, due to the spreading algorithm), they are known in the literature as Generalising Random Access Memories (GRAMs) [7].

A new spreading algorithm for GRAMs called *Combined Generalisation Algorithm* (CGA) was proposed in [11]. Such an algorithm combines the properties of weightless neural systems and those of weighted Hebbian learning systems [94]. A new metric measure called Discriminating Distance was described, which takes into consideration the discrimination power of the input variables by assigning a discrimination coefficient to each one of them. The discrimination coefficient is proportional to the input's ability to lead the node to fire, having a similar function to the synapses in the Hebbian learning [94].

GRAMs are often implemented as *Virtual* GRAMs (VGRAM) [79]. This is because the cost of storage in WNNs scales exponentially with the node fan-in (2^N). A VGRAM is a GRAM in which space is not allocated for patterns which are never found in the training phase. Instead, the system provides generalisation to unseen patterns during the use phase through calculation, based on the stored values and its spreading algorithm. However, VGRAMs can only be effective when an upper bound for the number of training patterns is known *a priori*, where such an upper bound has to be at least an order or two of magnitude less than the maximum capacity 2^N. Also, when using VGRAMs, the tradeoff between the gain in memory and the increase in response time, due to on-line calculations, has to be taken into account.

GRAMs have been often used in a recurrent architecture called GNUs, which will be described next.

6.1 GNU

The General Neural Unit (GNU) [12] is a single layer recurrent WNNs that uses GRAMs as nodes. The topology of GNUs differs from the classical Hopfield recurrent model [57] by having an internal feedback field as well as external feedforward connections. For instance, in [36] auto-associative (without external inputs) GNUs are shown to be able to learn reasoning structures found in semantic representation. In contrast, Sales *et al.* [95] carried out some linguistic experiments, concerning symbol grounding, by using heter-associative (with external inputs) GNUs.

In [70], auto-associative GNUs were shown to be able to store up to 2^N patterns and that given an initial vector, or initial state, the network retrieves the nearest stored one. Aleksander and Morton studied the retrieval properties of GNUs with external inputs and demonstrated that the optimum retrieval performance is obtained when there is perfect balance between the feedforward and feedback fields [12]. Braga[33], by using a probabilistic approach, obtained a set of expressions that enable the prediction of contradictions happening during the storage process of partially connected GNUs. The attractor properties of GNUs were also studied by Braga, who also demonstrated that the GNU's retrieval time is a function of the number of associations stored [32]. Recently, Adeodato & Taylor [2] developed another approach for the storage capacity assessment of WNNs which can be applied to different nodes and topologies, including GNUs. In such an approach, the probability of occurrence of collision is expressed as a function of the probability distribution of the training set.

The storage process in a GNU consists in creating an association between an external pattern and its internal representation by producing a re-entrant, or *stable*, state in which the network stabilises in the retrieval phase. The storage of an association $[\xi^e, \xi^i]$ between an external pattern ξ^e and its internal representation ξ^i is made by presenting ξ^e to the w external inputs and by clamping ξ^i at the node's outputs. In this situation, each site, or memory location, accessed in each one of the GRAM nodes should store a value (0 or 1) that makes this configuration stable in the retrieval phase. The value that should be stored in the site accessed in a given node j is the current value n_j of its output (corresponding to bit j of ξ^i), which creates a pointer to the state ξ^i itself when ξ^e is present at the inputs. Spreading occurs in each one of the GRAMs after all the input/output associations are presented. This procedure is similar to the one used to create fixed points in the state space of Hopfield networks [57]. Figure 7 illustrates the process of creating an association in GNUs.

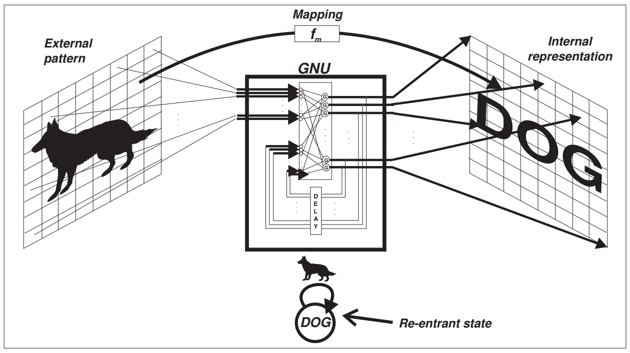

Figure 7: Creating an association in a GNU

During the retrieval phase, a GNU operates synchronously, with all the nodes updating their outputs at the same time, according to the sequence of values at their inputs. Due to the u-state locations left during the training process, the retrieval of associations has a stochastic nature, because some nodes will be flipping their outputs between 0 and 1, rather than generating a deterministic output for the current input vector. The state $S_j(t+1)$ of node j at time $t+1$ is a function of the sequence of values at its inputs at time t, as shown in Equation 4.

$$S_j(t+1) = F_j(t, An_0...An_{n-1}, Af_0...Af_{n-1}) \quad (4)$$

In Equation 4, $F_j(t, A_n, A_f)$ is the function performed by node j, An_i corresponds to the ith input sampled from the feedforward field and Af_i to the ith input sampled from the feedback field. The next state of the network is obtained by applying Equation 4 concurrently to every network node. Since the state of the node is a function of the values presented at both the feedforward and feedback fields, the successful retrieval of a known association $[\xi^e, \xi^i]$ after ξ^e is presented at the inputs depends also on the network's initial internal state.

In the extreme situation when the network is initialised in the target internal representation ξ^i, successful retrieval is guaranteed, but it may fail for other arbitrary initial states. The failure in the retrieval process is a consequence of learning associations beyond the network's capacity, which fills the state space with fixed points to which the network can be erroneously attracted. Successful retrieval of all the associations learned is expected if there is no overloading.

7 SDM

Kanerva [65] developed a method of associative storage, the Sparse Distributed Memory (SDM), that is physiologically plausible and has characteristics that make its hardware implementation attractive. Since such a model has many features similar to that of WNNs, in this section, the basic principle on which this system is based will be reviewed. A more detailed description of SDMs can be found in [65].

The essence of SDMs is to use sparsely distributed decoders in a high dimensional Boolean space so that any sparse decoder or *hard storage location* is accessed from anywhere in the space that is at a Hamming distance within r_p bits from its base address. Therefore, each decoder responds to all the vectors inside a hyper-sphere, or circle in SDM's terminology, with radius r_p and centre at the location's base address. Depending on the selected value for the radius r_p, input vectors can access more than one storage location at the same time, allowing data to be stored and retrieved concurrently to and from several memory storage locations.

A schematic view of an SDM is presented in Figure 8. Vector $\mathbf{d}$ in this figure contains the results of the computed distances between the input vector $\mathbf{I}$ and all the M sparse decoder's base addresses. Between vectors $\mathbf{d}$ and $\mathbf{y}$, threshold elements activate their outputs to 1 if the corresponding distances in vector $\mathbf{d}$ are within the range 0 to r_p. Therefore, the active elements of vector $\mathbf{y}$ are only those that correspond to the hard locations accessed by the input vector $\mathbf{I}$. Elements are stored in the matrix of hard locations $\mathbf{H}$ by the updating rule $\mathbf{H} = \sum \mathbf{y}(\mathbf{w})^T$, considering that both $\mathbf{y}$ and $\mathbf{w}$ are column vectors and that the binary vector $\mathbf{w}$ is represented with $+1$ and -1 elements. The retrieval of one stored vector, given the address vector I is accomplished by thresholding the output vector $\mathbf{v} = \mathbf{y}\mathbf{H}^T$ above 0.

SDMs can be seen as a general case of a RAM, since, in the limit case where r_p is set to zero and there are 2^n sparse decoders, an SDM actually becomes a conventional RAM.

The main differences between a RAM and an SDM are:

- The address decoder of a RAM is selected only if its base address is presented at the inputs ($r_p = 0$), whereas a sparse decoder of an SDM is selected from anywhere within r_p bits Hamming distance to the base address ($r_p > 0$);
- A conventional RAM has 2^n decoders and storage locations, where n is the size of the input address vector, whereas an SDM has only a portion M of the maximum 2^n sparse decoders and hard storage locations;
- A hard storage location of an SDM, instead of storing only one data vector, like the storage locations of a RAM, keeps a statistical measure of all the data vectors that accessed that location during the storage process.

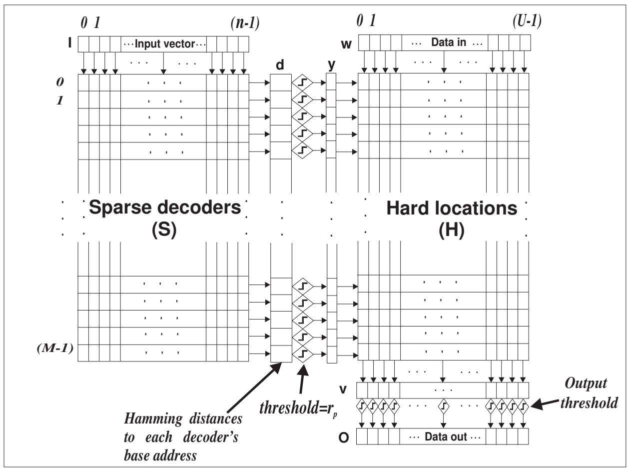

Figure 8: Schematic view of the Sparse Distributed Memory (SDM)

8 Comparative Studies

Two types of comparative study associating weighted and weightless networks have been presented in the literature. The first focuses on learning, generalisation, content addressable systems, and hardware implementation. The second was presented in [9]. Such a study was directed specifically at the Parallel Distributed Processing (PDP) framework [94]. In these comparative studies, the details of the weightless systems are given while many details of the weighted are omitted for being well known. The main results of the comparison are listed below:

- **Learning**: In contrast to the weighted neural models, there are several one-shot learning algorithms for WNNs, where training takes only one epoch [12, 42, 47, 90]. Also, like for the weighted neural systems, there exist many iterative learning schemes for WNNs. For instance, in WNNs the tasks carried out by error back-propagation [93]can be achieved by feeding error information to all the hidden layers of a feedforward system in parallel. A training algorithm for this task can be found in [3]. Implementations of Kohonen maps [68] with weightless nodes can be found in [15, 38, 85]. Learning schemes similar to the Adaptive Resonance Theory [35], implemented with WNNs, were presented in [47, 67]. Reinforcement learning procedures [25] for the WNNs were introduced in [52, 84, 81]. Additionally, there are algorithms based on standard statistical methods [24, 31, 72, 98, 92], where some of them, such as the ones in [31, 98], are peculiar to the weightless neural models.

- **Generalisation**: generalisation in WNNs is mainly obtained from the network as a whole. A conventional RAM node by itself does not generalise. That is, in WNNs, generalisation is sensitive to similarities to entire patterns instead of their constituent variables [8]. Thus, generalisation in WNNs is affected mostly by the node fan-in and the connectivity pattern of the network. In contrast, in the weighted neural models, individual neurons can be sensitive to particular input variables. Nevertheless, generalisation at the node level can also be obtained with some weightless nodes such as GRAM [7] and pRAM [51].

- **Content addressable systems**: WNNs with the same kind of configuration of Hopfield networks, that is, fully connected auto-associative networks, were analysed by Lucy in [70]. Lucy's work showed that the storage capacity of fully connected auto-associative GNUs grows exponentially with the number k of nodes, that is, $O(2^k)$. In contrast, in Hopfield networks such a capacity scales only linearly with the number of nodes, that is, $O(k/\ln(k))$ [76]. However, the cost of storage in GNUs also grows exponentially with the number of nodes. Differently, in Hopfield networks the cost of storage is only $O(k^2)$.

Nevertheless, weightless networks such as GNUs do not need to be fully connected as Hopfield networks. In [2, 32], two different methods to design partially connected GNUs with high confidence in the storage process are proposed. In [32], it was also shown that partially connected GNUs could have better retrieval performance than the fully connected ones, provided that the training set size is small enough to avoid saturation and overwriting in the storage process.

- **Issues on hardware implementation**: The possibility of implementing real networks in hardware is often considered to be one of the main features of WNNs. However, this is mainly the case for simpler neurons such as the RAM node, which can be easily implemented by using commercially available RAM components. In general, nodes such as GRAMs, which require the spreading of information to neighbouring memory locations, are not straightforwardly implementable in hardware such as the WISARD system. So, they might require an amount of circuitry comparable to that of weighted neural systems. In [14, 41], respectively, examples of hardware implementation of the WISARD system (discrete components) and pRAM networks (VLSI) are presented, as well as their learning algorithm. In [96], a technique for hardware and software implementation of trained RAM and GSN networks is introduced. Kennedy *et al.* [66] described the **SAT** processor; a dedicated hardware implementation of a binary neural image processor. The **SAT** processor is aimed specifically at supporting the ADAM networks.

Now, definitions of all the major concepts concerned with the PDP framework (set of processing units, state of activation, output functions, pattern of connectivity, propagation rule, learning rule and environment) will be given to WNNs. These definitions are briefly described below for WNNs [9]:

1. **Set of processing units**: the weightless nodes.
2. **State of activation**: a vector of N elements where each element is the decimal representation of the input of the correspondent node.
3. **Output of the units**: the output r of the node as described in Equations 1 and 3 (or the identity function for RAM nodes).
4. **Pattern of connectivity**: binary matrix without adaptive weights.
5. **Rule of propagation**: the responsibility for learning lies in the input-output mappings found in the units themselves.
6. **Activation rule**: a mapping from the input into the output of the net.
7. **Learning rule**: changes in the contents of the truth table.

In [61] is presented a comparative study regarding the information capacity of two models with binary weights: the Willshaw model and the Inverted Neural Network.

9 Future Work and Research Issues

As in the neurocomputing field in general, there are also two salient directions for research in weightless systems: new insights into biological brains and the design of more efficient computing machinery. Weightless does not imply distance from biological reality. On the contrary, the methodology has more flexible modelling capabilities which enables it to accommodate new discoveries in living systems [39, 81]. Some fundamental issues will be considered below:

1. **The Scaling Problem**. Most of the neural configurations (autoassociators in particular) have the problem of poor growth of storage capacity with the size of the system. On the other hand, cognitive computing needs efficient storage for "experience". To solve this problem, ways to decompose "experience" need to be considered. Studies about storage capacity of RAM-based networks are presented in [1, 33, 31, 86, 98].

2. **Planning**. One of the major differences between rule-based and neural computing is the notion that a neural net can find the solution for a planning problem directly, whereas rule-based systems have to search quasi-exhaustively. Planning is a somewhat neglected area of neurocomputing. Learning by exploration, generalisation to new cases and the use of several levels of abstraction simultaneously are promising areas for research. Mrsic-Flogel ([79]) described a weightless system for planning. In [20], a novel symbolic reasoning system based upon ADAM is presented.

3. **Temporal Computation**. Many aspects of temporal processing demand attention. Ludermir ([71]), developed weightless techniques for the recognition of the syntactic structure of sequences. She also developed algorithms using cut-point nodes which generate production rules for weightless neural networks and generate weightless neural networks for production rules. Recently, such a work was extended for the case of single-layer pRAM networks [46]. But work needs to be done on semantics. In order to achieve the appropriate balance between "understanding" of language imparted by syntax and that imparted by semantics, the linguistic behaviour of temporal neural structures needs to be studied. Sales *et al.* [95] present a neural state machine, constructed from WNNs, to "ground" visual and linguistic representation. An interesting analysis of the dynamic of temporal neural structures can be found in [77, 103, 105, 107].

4. **Applications**. There are many applications, such as scene and language understanding, where conventional computing fails to provide efficient solutions, and where neurocomputing offers the potential of making computational machinery more competent. The ADAM network has been used in scene analysis applications [19]. In [87], a pRAM network for the classification of objects in a video sequence of FLIR (forward looking infra-red) images is described.

10 Conclusions

It has been shown in the paper that WNN models define an important paradigm in the field of Artificial Neural Networks, by providing flexibility, fast learning algorithms and easily implementable nodes. A great deal of work has been done in the field since the N-tuple sampling technique was proposed in the late 50s and much has still to be done. The basic RAM node has improved by the addition of new features and learning schemes, resulting in neural models such as PLNs, GRAMs, GSNs, pRAMs, etc. The features of these recently described WNN models support the weightless approach as very promising to overcome the challenges in the field of Artificial Neural Networks. A comprehensive collection of recent papers on RAM-based networks can be found in [21].

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments and valuable suggestions.

References

[1] P. J. L. Adeodato and J. Taylor. Storage capacity of RAM-based neural networks: pyramids. *Neural Networks World*, 6(3):241–249, 1996.

[2] P. J. L. Adeodato and J. G. Taylor. Analysis of the storage capacity of RAM-based neural networks. In D. L. Bisset, editor, *Proceedings of the Weightless Neural Network Workshop (WNNW95)*, pages 103–110, University of Kant at Canterbury, UK, 1995.

[3] R. Al-Alawi and T. J. Stonham. A training strategy and functionality analisys of digital multi-layer neural networks. *Journal of Intelligent Systems*, 2:53–93, 1992.

[4] I. Aleksander. Self-adaptive universal logic circuits. *IEE Electronics Letters*, 2:231, 1966.

[5] I. Aleksander. Emergent intelligent properties of progressively structured pattern recognition nets. *Pattern Recognition Letters*, 1:375–384, 1983.

[6] I. Aleksander. Canonical neural nets based on logic nodes. In *Proceedings of the IEE International Conference on Artificial Neural Network*, pages 110–114, London, 1989.

[7] I. Aleksander. Ideal neurons for neural computers. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 225–228. Elsevier Science, 1989.

[8] I. Aleksander. *An introduction to neural computing.* Chapman and Hall, London, 1990.

[9] I. Aleksander. Connectionism or weightless neurocomputing. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Proceedings of International Conference on Artificial Neural Networks (ICANN91)*, volume 2, pages 991–1000, Helsinki, 1991.

[10] I. Aleksander and R. C. Albrow. Pattern recognition with adaptive logic elements. In *Proceedings of IEE-NPL Conference Pattern Recognition*, 1968.

[11] I. Aleksander, T. J. Clarke, and A. P. Braga. Binary neural systems: combining weighted and weightless properties. *IEE Intelligent Systems Engineering*, 3:211–221, 1994.

[12] I. Aleksander and H. Morton. General neural unit: retrieval performance. *IEE Electronics Letters*, 27:1776–1778, 1991.

[13] I. Aleksander and T. J. Stonham. Guide to pattern recognition using random-acess memories. *Computers and Digital Techniques*, 2:29–40, 1979.

[14] I. Aleksander, W. V Thomas, and P. A. Bowden. WISARD: a radical step forward in image recognition. *Sensor Review*, 4(3):120–124, 1984.

[15] N. M. Allison and M. J. Johnson. Realisation of self-organising neural maps in $\{0,1\}^n$-space. In J. G. Taylor and C. L. T. Mannion, editors, *New Developments in Neural Computing*, pages 79–86. IOP Publishing Ltd, 1989.

[16] J. Austin. Grey scale N-tuple processing. In J. Kittler, editor, *Proceedings of the IV International Conference on Pattern Recognition*, pages 110–119, Cambridge, 1988.

[17] J. Austin. A review of the advanced Distributed Associative Memory (ADAM). In *Proceedings of the Weightless Neural Network Workshop (WNNW93)*, pages 24–28, University of York, York, UK, 1993.

[18] J. Austin A Review of RAM-based Neural Networks. In *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 58–66, 1994. IEEE Computer Society Press.

[19] J. Austin. Associative memories and application of neural networks to vision. In R. Beale, editor, *Handbook of Neural Computation.* Oxford University Press, 1995.

[20] J. Austin. Distributed associative memories for high-speed symbolic reasoning. *Fuzzy Sets and Systems*, 82(2):223–233, 1996.

[21] J. Austin. *RAM-Based Neural Networks.* World Scientific, York, UK, 1998.

[22] J. Austin. RAM-Based Neural Networks, a Short History. In J. Austim, editor, *RAM-Based Neural Networks*, pages 3–17, York, UK, 1998.

[23] J. Austin and T. J. Stonham. An associative memory for use in image recognition and occlusion analysis. *Image and Vision Computing*, 5:251–261, 1987.

[24] A. Badr. N-tuple classifier for ECG signals. In N. M. Allinson, editor, *Proceedings of the Weightless Neural Network Workshop (WNNW93)*, pages 29–32, University of York, York, UK, 1993.

[25] A. G. Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiol.*, 4:229–256, 1985.

[26] J. M. Bishop, P. R. Minchinton, and R. J. Mitchell. Real time invariant grey level image processing using digital neural networks. In *Proceedings of IMechE Conference, EuroTech Direct'91 - Computers in the Engineering Industry*, pages 187–199, Birmingham, 1991.

[27] W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of Eastern Joint Computer Conference*, pages 225–232, Boston, 1959.

[28] W. W. Bledsoe and C. L. Bisson. Improved memory matrices for the n-tuple recognition method. In *Proceedings of IRE Joint Computer Conference,11*, pages 414–415, 1962.

[29] G. Bolt. Fault tolerance in binary neural networks. In *Proceedings of Weightless Neural Network Workshop (WNNW93)*, pages 64–69, University of York, York, UK, 1993.

[30] R. G. Bowmaker and G. G. Coghill. Improved recognition capabilities for goal seeking neuron. *IEE Electronics Letters*, 28:220–221, 1992.

[31] N. P. Bradshaw. Improving the generalisation of the N-tuple classifier using the effective VC dimension. *IEE Electronics Letters*, 32(20):1904–1905, 1996.

[32] A. P. Braga. Attractor properties of recurrent networks with generalising Boolean nodes. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN94)*, pages 421–424. Springer-Verlag, 1994.

[33] A. P. Braga. Predicting contradictions in the storage process of diluted recurrent Boolean neural networks. *IEE Electronics Letters*, 30:55–56, 1994.

[34] A. P. Braga and I. Aleksander. Geometrical treatment and statistical modelling of the the distribution of patterns in the n-dimensional Boolean space. *Pattern Recognition Letters*, 16:507–515, 1994.

[35] G. A. Carpenter and S. Grossberg. Adaptive resonance theory: neural networks architectures for self-organizing pattern recognition. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Proceedings of Parallel Processing in Neural Systems and Computers*, 1990.

[36] J. Castro. Semantic knowledge in general neural units: issues of representation. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 674–679, Australia, 1995.

[37] S. S. Christensen, A. W. Andersen, T. M. Jorgensen, and C. Liisberg. Visual guidance of a pig evisceration robot using neural networks. *Pattern Recognition Letters*, 17(4):345–355, 1996.

[38] T. G. Clarkson, D. Gorse, and J. G. Taylor. Applications of the pRAM. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN91)*, pages 2618–2623, Singapore, 1991.

[39] T. G. Clarkson, D. Gorse, and J. G. Taylor. Biologically plausible learning in hardware realiziable nets. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN91)*, pages 195–199, Helsinki, Finland, 1991.

[40] T. G. Clarkson, Y. Guan, and J. G. Taylor. Generalization in probabilistic RAM nets. *IEEE Transcactions on Computers*, 4(2):360–363, 1993.

[41] T. G. Clarkson, C. K. Ng, D. Gorse, and J. G. Taylor. Learning probabilistic RAM nets using VLSI structures. *IEEE Transactions on Computers*, 41(12):1552–1561, 1992.

[42] A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. SOFT - A Boolean self organising feature extractor. In I. Aleksander and J. G. Taylor, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN92)*, pages 669–672, Brighton, UK, September 1992.

[43] A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. Progressive learning algorithm for GSN feedforward neural architectures. *IEE Electronics Letters*, 30(6):506–507, March 1994.

[44] A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. Combining Boolean Neural Architectures for Image Recognition. *Connection Science*, 9(4):506–507, December 1997.

[45] M. C. P. de Souto, K. S. Guimarães, and T. B Ludermir. On the intractability of loading pyramidal architectures. In *Proceedings of the IEE International Conference on Artificial Neural Networks*, pages 189–194, UK, 1995.

[46] M. C. P. de Souto, T. B. Ludermir, and W. R. de Oliveira. Synthesis of probabilistic automata in pRAM neural networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN98)*, pages 603–608, Sweden, 1998.

[47] E. Filho, M. C. Fairhurst, and D. L. Bisset. Adaptive pattern recognition using goal-seeking neurons. *Pattern Recognition Letters*, 12:131–138, March 1991.

[48] D. Gorse, D. A. Romano-Critchley, and J. G. Taylor. A pulse-based reinforcement algorithm for learning continuous functions. *Neurocomputing*, 14(4):319–344, 1997.

[49] D. Gorse and J. G. Taylor. On the equivalence properties of noisy neural and probabilistic RAM nets. *Physics Letters A*, 131(6):326–332, 1988.

[50] D. Gorse and J. G. Taylor. Reinforcement training strategies for probabilistic RAMs. In M. Novak and E. Pelikan, editors, *Proceedings of the International Symposium on Neural Networks and Neurocomputing (NEURONET90)*, pages 180–184, 1990.

[51] D. Gorse and J. G. Taylor. A continuous input RAM-based stochastic neural model. *Neural Networks*, 4:657–665, 1991.

[52] D. Gorse and J. G. Taylor. Universal associative stochastic learning automata. In *Neural Networks World 1*, pages 193–202, 1991.

[53] D. Gorse and J. G. Taylor. Review of the theory of pRAMs. In *Proceedings of the Weightless Neural Network Workshop (WNNW93)*, University of York, UK, 1993.

[54] Y. Guan, T. G. Clarkson, D. Gorse, and J. G. Taylor. Noisy reinforcement training for pRAM nets. *Neural Networks*, 7(3):523–538, 1994.

[55] K N. Gurney. Training nets of hardware realizable sigma-pi units. *Neural Networks*, 5:289–303, 1992.

[56] L. Hepplewhite and T. J. Stonham. N-tuple texture recognition and the zero crossing sketch. *IEE Electronics Letters*, 33(1):45–46, 1997.

[57] J. J. Hopfield. Neural networks and physical systems with emergent collective computational properties. In *Proceedings of the National Academy of Sciences (USA)*, volume 79, pages 2554–2558, 1982.

[58] G. Howells, M. Fairhurst, and D. Bisset. BCN: A novel network architecture for RAM-based neurons. *Pattern Recognition Letters*, 16:297–303, March 1995.

[59] G. Howells, M. C. Fairhurst, and D. L. Bisset. GCN: The Generalised Convergent Network. In *Proceedings of the IEE International Conference on Image Processing and its Application*, 1995.

[60] G. Howells, M. C. Fairhurst, and D. L. Bisset. PCN: The Probabilistic Convergent Network. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN95)*, pages 1211–1214, Perth, Australia, November 1995.

[61] A. Jagota, G. Narasimhan, and K. Regan. Information Capacity of Binary Weights Associative Memories. *Neurocomputing*, 19(1-3), April 1998.

[62] T. M. Jorgensen. Classification of handwritten digits using a RAM neural net architecture. *International Journal of Neural Systems*, 8(1):17–25, 1997.

[63] T. M. Jorgensen, S. S. Christensen, and C. Liisberg. Crossvalidation and information measures for RAM based neural networks. In D. Bisset, editor, *Proceedings of the Weightless Neural Networks Workshop (WNNW95)*, pages 87–92, University of Kent at Canterbury, UK, 1995.

[64] W. K Kan and I. Aleksander. A probabilistic logic neuron network for associative learning. In *Proceedings of the IEEE First International Conference on Neural Networks (ICNN87)*, volume II, pages 541–548, San Diego, California, June 1987.

[65] P. Kanerva. *Sparse Distributed Memory.* MIT Press, 1988.

[66] J. V. Kennedy and J. Austin. A hardware implementation of a binary neural image processor. In *Proceedings of the IV International Conference on Microeletronics for Neural Networks and Fuzzy Systems*, pages 178–185, Torino. Italy, 1994.

[67] M. A. Kerin and T. J. Stonham. Face recognition using digital neural network with self-organising capabilities. In *Proceedings of the X International Conference on Pattern Recognition*, pages 738–741, 1990.

[68] T. Kohonen. *Self-Organising and Associative Memory.* Springer-Verlag, Heidelberg, 1988.

[69] A. Kolcz and N. M. Allinson. Application of the CMAC input encoding scheme in the N-tuple approximation network. *IEE Proceedings-E Computers and Digital Techniques*, 141(3):177–183, 1994.

[70] J. Lucy. Perfect auto-associators using RAM-type nodes. *IEE Electronics Letters*, 27:799–800, 1991.

[71] T. B. Ludermir. Computability of logical neural networks. *Journal of Intelligent Systems*, 2:261–290, 1992.

[72] S. P. Luttrell. Gibbs distribution theory of adaptive n-tuple networks. In I. Aleksander and J. Taylor, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN92)*, pages 313–316, Brighton, UK, 1992.

[73] W. Martins and N. M. Allinson. Two improvements for GSN neural networks. In N. M. Allison, editor, *Proceedings of the Weightless Neural Network Workshop (WNNW93)*, pages 58–63, University of York, York, UK, 1993.

[74] J. D. McCauley, B. R. Thabe, and A. D. Whittaker. Fat estimation in beef ultrasound images using texture and adaptive logic networks. *Transactions of ASAE*, 37(3):997–102, June 1994.

[75] W. S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Imminent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1930.

[76] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33:461–482, 1987.

[77] D. K. Milligan and M. J. D. Wilson. The behaviour of affine Boolean sequential networks. *Connection Science*, 5(2):153–167, 1993.

[78] R. J. Mitchell, J. M. Bishop, and P. R. Minchinton. Optimizing memory usage in N-tuple neural networks. *Mathematics and Computers in Simulation*, 40(5-6):549–563, 1996.

[79] J. Mrsic-Flogel. Approaching cognitive system design. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN91)*, volume 1, pages 879–883, Helsinki, 1991.

[80] S. Muroga. Lower bounds of the number of threshold functions and a maximum weight. *IEEE Transactions on Eletronic Computers*, pages 136–148, 1965.

[81] C. Myers. *Delay learning in artificial neural networks*. Chapman & Hall, 1992.

[82] C. Myers and I. Aleksander. Learning algorithms for probabilistic logic nodes. In *Abstracts of the I Annual International Neural Networks Society Meeting (INNS88)*, page 205, Boston, 1988.

[83] C. Myers and I. Aleksander. Output functions for probabilistic logic nodes. In *Proceedings of the IEE International Conference on Neural Networks*, pages 310–314, London, 1989.

[84] R. S. Neville and T. J. Stonham. Adaptive reward-penalty for probabilistic logic nodes. In I. Aleksander and J. G. Taylor, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN92)*, volume 1, pages 631–634, Brighton, UK, 1992.

[85] P. Ntourntoufis. Self-organisation properties of a discriminator-based neural network. In M. Caudill, editor, *Proceedings of the International Joint Conference on Neural Networks (IJCNN90)*, volume 2, pages 319–324, Washington, USA, 1990.

[86] W. Penny and T. J. Stonham. Storage capacity of multilayer Boolean neural networks. *IEE Electronics Letters*, 29:1340–1341, 1993.

[87] S. Ramanan, R. S. Petersen, T. G. Clarkson, and J. G. Taylor. pRAM nets for detection of small targets in sequence of infra-red images. *Neural Networks*, pages 1227–1237, 1995.

[88] R. Rohwer and D. Cressy. Phoneme classification by Boolean networks. In J. P. Tubach and J. J. Mariani, editors, *Proceedings of the European Conference on Speech Communication and Technology*, pages 557–560, Edinburgh, 1989.

[89] R. Rohwer and A. Lamb. An exploration of the effect of super large n-tuple on single-layer RAMnets. In N. M. Allison, editor, *Proceedings of the Weightless Neural Network Workshop (WNNW93)*, pages 33–37, Univeristy of York, York, UK, 1993.

[90] R. Rohwer and M. Morciniec. A theoretical and experimental account of n-tuple classifier performance. *Neural Computing*, 8:629–642, 1995.

[91] R. Rohwer and M. Morciniec. The theoretical and experimental status of the n-tuple classifier. *Neural Networks*, 11(1):1–14, 1998.

[92] R. J. Rohwer. Two bayesian treatments of the n-tuple recognition method. In *Proceedings of the IEE International Conference on Neural Networks*, pages 171–176, UK, 1995.

[93] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[94] D. E. Rumelhart and J. L.; McClelland. *Parallel Distributed Processing*, volume 1: Foundations. The MIT Press, 1986.

[95] N. Sales, R. Evans, and I. Aleksander. Succesful naive representation grounding. *Artificial Intelligence Review*, 10(1-2):83–102, 1996.

[96] E. V. Simões, L. F. Uebel, and D. A. C. Barone. Hardware implementation of RAM neural networks. *Pattern Recognition Letters*, 17(4):421–429, 1996.

[97] M. J. Sixsmith, G. D. Tattersall, and J. M. Rollet. Speech recognition using n-tuple techniques. *British Telecom Technology Journal*, 8(2):50–60, 1990.

[98] R. Tarling and R. Rohwer. Efficient use of training data in the n-tuple recognition method. *IEE Electronics Letters*, 29(24):2093–2094, 1993.

[99] J. G. Taylor. Spontaneous behaviour in neural networks. *Journal of Theoretical Biology*, 36:513–528, 1972.

[100] J. R. Ullman. Experiments with the n-tuple method of pattern recognition. *IEEE Transactions on computers*, pages 1135–1137, 1969.

[101] J. R. Ullman and P. A. Kidd. Recognition experiments with typed numeral from envelopes in the mail. *Pattern Recognition*, (1):273–289, 1969.

[102] V. N. Vapnik and A. Ja Chervonenkis. The necessary and sufficient conditions for consistency of the method of empical risk minimisation. *Pattern Recognition and Image Analysis*, 1(3):284–305, 1991.

[103] G. Vasconcelos and C. Fernandes. Error recovery behaviour of feedback RAM-networks. In *Proceedings of the IEE International Conference on Artificial Neural Networks*, volume 349, pages 304–308, UK, 1991.

[104] Y. S. Wang, B. J. Griffiths, and B. A. Wilkie. A novel system for coloured object recognition. *Computers in Industry*, 32(1):69–77, 1996.

[105] T. L. H. Watkin and D. Sherrington. Sequences of memories in Boolean networks. *Physica A*, 185:449–452, 1992.

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

[107] M. J. D. Wilson and D. K. Milligan. Cyclic behaviour of autonomous, synchronous Boolean networks: some theorems and conjectures. *Connection Science*, 4(2):143–154, 1992.

[108] B. Zhang, L. Zhang, and H. Zhang. The complexity of learning in PLN networks. *Neural Networks*, 8(2):221–228, 1995.