Automata and Weightless Neural Networks

Teresa B. Ludermir* Wilson R. de Oliveira†

Departamento de Informática

Universidade Federal de Pernambuco

Recife - PE - Brazil

CEP: 50.739

e-mails: tbl@di.ufpe.br wrdo@di.ufpe.br

Abstract

The aim of this paper is to compare Automata with Weightless Neural Networks (WNN). The WNN of Aleksander ([5]) is extended initially to compute Weighted Regular Languages and then to simulate the behaviour of a Turing Machine.

1 Introduction

A Weightless neural network is an arrangement of a finite number of nodes in any number of layers, with or without feedback connections between the nodes and where the nodes are RAM nodes storing n-bit words ([4]). The n-bits have been considered to store a number in the interval from 0 to 1 which represents the 'firing probability' of the node. There are many advantages of such models in relation to weighted-sum-and-threshold approaches and the most important one is that this model is straightforward to implement in hardware.

*Partially supported by Nuffield Foundation and FACEPE (Research Foundation of the State of Pernambuco).

†Supported by the Brazilian Research Council (CNPq).

Natural networks can easily process temporal information whilst artificial networks to date are not well-suited for dealing with time-varying input patterns. Different recurrent neural networks models have been used to learn temporal sequences, for example see ([16]). Most of these systems simulate the behaviour of finite state machines. They are able to learn some regular languages. Regular languages are generated by regular grammars and they are the simplest languages in the Chomsky hierarchy ([12]).

Some basic principles of Automata Theory are necessary for the understanding of this work. These principles are the concepts of probabilist automata ([26]), weighted regular grammars and languages ([27]) and the Chomsky's hierarchy ([12]). These concepts are omitted here but they can be found in the literature mentioned above. One important result is the relationship between the Chomsky's hierarchy and weighted regular languages (these are the languages recognised by probabilist automata) which can be illustrated by the following equations: (1) WRL $\subset$ RL; (2) WRL $\cap$ CFL $\neq \phi$; (3) WRL $\cap$ CSL $\neq \phi$; (4) WRL $\cap$ TZL $\neq \phi$; where the abbreviated notation means: WRL is the set of weighted regular languages, RL is the set of regular languages, CFL is the set of context-free languages, CSL is the set of context-sensitive languages and TZL is the set of type zero languages.

2 RAM, PLN, MPLN

Definition 1 A RAM Neural Network is an arrangement of a finite number of neurons in any number of layers, in which the neurons are RAM (Random Access Memory) nodes. The RAM node is represented in the Figure 1 below:

The input may represent external input or the output of neurons from another layer or a feedback input. The data out may be 0's or 1's. The set of connections is fixed and there are no weights in such nets. Instead the function performed by the neuron is determined by the contents of the RAM - its output is the value accessed by the activated memory location. There are 2^{2^N} different functions which can be performed on N address lines and these correspond exactly to the 2^N states that the RAM can be in, that is a single RAM

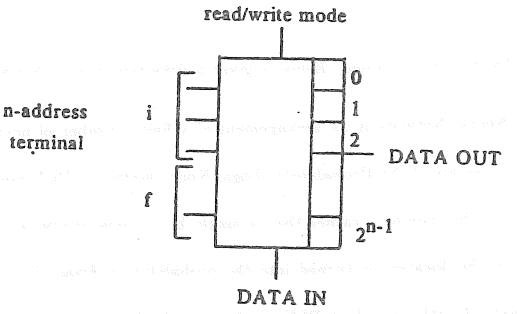

Figure 1: RAM node

can compute any function of its inputs.

Seeing a RAM node as look-up table (truth table) the output of the RAM node is described by equation 1 below:

$$r = \begin{cases} 0 & \text{se } C[p] = 0 \\ 1 & \text{se } C[p] = 1 \end{cases} \quad (1)$$

where $C[p]$ is the content of the address position associated with the input pattern p.

Learning in a RAM node takes place simply by writing into it, which is much simpler than the adjustment of weights as in a weighted-sum-and-threshold network. 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 function of its input. There is no generalisation in the RAM node itself (the node must store the appropriate response for every possible input), but there is generalisation in networks composed of RAM nodes (Aleksander [2]). Generalisation in weightless networks is affected first by the diversity of the patterns in the training set, that is, the more diverse the patterns in the training set, the greater will be the number of subpatterns seen by each RAM, resulting in a larger generalisation set. Secondly, the connection of RAMs to common features in the training set reduces the generalisation set.

The introduction of a probabilistic element into the weightless node was proposed by Aleksander (Aleksander [4]) after attempting a rapprochement between Boltzmann machines and weightless node networks (Aleksander [3]). He called the node with this probabilistic element a probabilistic logic node (PLN). The main feature of the PLN model is the unknown state, u, which responds with a randomly generated output

for inputs on which it has not been trained. Below is given a definition of a PLN node.

Definition 2 A PLN Neural Network is an arrangement of a finite number of neurons in any number of layers, in which the neurons are PLN (Probabilistic Logic Node) nodes. A PLN node differs from a RAM node in the sense 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 a 1) at the overall output of the node. In other words, a PLN consists of a RAM-node augmented with a probabilistic output generator. As in a simple RAM node, the I binary inputs to a node form an address into one of the 2^I RAM locations. Simple RAM nodes then output the stored value directly. In the PLN, the value stored at this address is passed through the output function which converts it into a binary node output. Thus each stored value may be interpreted as affecting the probability of outputting a 1 for a given pattern of node inputs.

The contents of the nodes are 0's, 1's and u's. u means the same probability to produce as output 0 or 1. The output of the PLN node is described by equation 2 below:

$$r = \begin{cases} 0 & \text{se } C[p] = 0 \\ 1 & \text{se } C[p] = 1 \\ \text{random}(0,1) & \text{se } C[p] = u \end{cases} \qquad (2)$$

where $C[p]$ is the content of address position associated with the input pattern p and random(0,1) is a random function that generates zeros and ones with the same probability.

Training with 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 start of training, all stored values in all nodes are initialised to u, and thus the net's behaviour is completely unbiased. In a fully converged PLN net, every addressed location should contain a 0 or a 1, and the net's behaviour will be completely deterministic. [1]

[1] There may be PLN locations which are never addressed, e.g. addresses to nodes in the input layer which represent n-tuples not present in the training set, or to nodes in higher layers if some combinations of lower node outputs never occur. These unaddressed locations may contain u without affecting the status of the net as converged.

Weightless PLN nodes also have had notable success in a number of applications. PLN networks have some advantages relative to RAM networks. First, because there are no pre-existing structures in the state space of PLN nets, they are easier to train (Kan-Aleksander [13]). There are a number of training strategies for PLN networks such as the ones in ((Aleksander-Morton [5]), (Myers [22]) and (Al-Alawi-Stonham [1])). Second, while RAM nets are not very sensitive to small differences in input patterns, PLN nets can be made very sensitive if they are organised into a pyramid of PLNs (Aleksander-Morton [5]). Thirdly, there are various sources of noise in the activity of natural neurons in neural networks (Taylor [30]); and with the stochastic activity of PLNs a slightly more realistic modeling of neural activity is achieved than with RAMs. Forthly, by experimental results, it is known that when solving the same problem with PLN nets and RAM nets, it is possible in many cases to save states when using PLN nets. This means that smaller number of nodes are necessary when using PLN networks than when using RAM networks.

The Multiple-valued Probabilistic Logic Neuron (MPLN) (Myers [23]) is simply an extension of the PLN. A MPLN differs from a PLN node in the sense that a q-bit number (rather than two bits) is now stored at the addressed memory location. The content of this location is also the probability of firing at the overall output of the node. Say that q is 3, then the numbers 0 to 7 can be stored in each location of the MPLN. One way of regarding the actual number stored may be as a direct representation of the firing probability by treating the number as a fraction of 7. So a stored 2 would cause the output to fire with a probability of 2/7, and so on.

One result of extending the PLN to MPLN is that the node locations may now store output probabilities which are more finely gradated than in the PLN - for example, it is possible that a node will output 1 with 3% probability under a certain input. The second result of the extension is that learning can allow incremental changes in stored values. In 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 indicate its validity.

There are three important parameters involved in the MPLN models:

- the number of inputs of the node I (and hence 2^I, the number of stored values in the node): I has a direct influence on the memory requirements and on the tradeoff between generalisation and memorisation (this is also true for RAM, PLN and most of the weightless nodes).
- the number of elements representing possible stored values which is defined by the number of bits used in the nodes: the number of elements used to represent the probability of outputs will influence the speed of learning.
- the output function which is applied on the addressed content: the output function may be probabilistic, linear, step or sigmoid function as described for the weighted node.

3 Description of the new node

In this section the new node ([18]) is explained. This new node is an extension of the MPLN node and it is based on ideas from the theory of probabilistic automata. Probabilistic automata recognise patterns depend on two information. The final state of the automaton after the pattern has been submitted to it and the the overall probability associated with such pattern. If the final state of the automaton belongs to the set of final states of the language to be recognised, then the probability associated with the pattern is compared with the threshold associated with the language. Only if the probability is bigger than the threshold, the pattern is considered accepted by the automaton as belonging to the language in question.

To simulate the behaviour of a probabilistic automata with neural networks we need to have a node capable of storing the probability of the path followed by the network up to that node with a fed pattern. To be able to store the probability of the path extra memory was given to our node. That is, besides the n words of B-bits (considering a node with n inputs) the node is going to have an extra word with k-bits to store the probability.

With a MPLN node the numbers stored in the n words of B-bits represent the 'firing probability' of the neuron and the node outputs 1/0 depending on this 'firing probability'. Once the node outputted 1/0 there is no other use for the 'firing probability' with a MPLN. The overall output of the network is only a pattern.

without any probability. In our case, the overall output of the network needs to be the pattern followed by the probability associated with this pattern. The way the probability is calculate depends on the type of the function the node is computing and it will be explained with more details later in this paper. Every node needs to have only one probability associated with it in order to calculate the probability of the whole pattern submitted to the network. This probability is used basically in the recognition phase of the system.

4 Description of the recognition method

A different method of achieving pattern recognition is used with our model. This method was firstly used for MPLN networks in ([17]), and we called it cut-point recognition algorithm. The output of our network consists of two parts: a pattern (which is the final state of the network) and a probability (which is the probability of the input to be recognised by the network). Patterns will be recognised by the network if the last states of the network, when such patterns were submitted to it, are in the set of final states of the network for the language in question and the probabilities associated with the patterns are greater than the threshold.

A short description of the cut point recognition algorithm, applied for the new nodes, is given below, where P is the probability of the path in each node and pf is the probability of the node to fire. ALGORITHM

1. Make for all nodes $P = 0$ (there is no path followed before the first symbol of each pattern is fed).

2. Choose an input pattern $\tilde{X} = X_1X_2...X_n$.

3. Feed the pattern $\tilde{X} = X_1X_2...X_n$ into the network symbol by symbol and calculate the overall probability associated with the pattern $\tilde{X}$.

   For every symbol X_i do

   3.1 Feed symbol X_i to all external input terminals.

   3.2 For all nodes j in the network do

If the node j fire then calculate probability P for this node.

3.2.1 If the node j is an *and* node $P = pf_j * P_{inputs\ of\ the\ node\ j\ that\ have\ fire\ in\ time\ i}$.

3.2.2 If the node j is an *or* node $P = \sum P_{inputs\ of\ the\ node\ j\ that\ have\ fire\ in\ time\ i}$.

3.2.3 If the node j is a *complement* or a *delay* node

$P = P_{input\ of\ the\ node\ j\ if\ it\ has\ fired\ in\ time\ i}$

4. If the final state of the net is in the set of final states for the language then compare the overall probability with the threshold. If the probability is bigger than the threshold then the pattern $\tilde{X}$ is accepted as belonging to the language.

This algorithm was based on the way probabilistic automata recognise patterns with thresholds. The class of patterns recognised by the network depends not only on the structure of the network but also on the threshold associated with the network. This algorithm working together with the new node not only increases the computation power of the network but also it is more appropriate to pattern recognition because it gives a deterministic decision as to whether a pattern $\tilde{X}$ belongs to the language recognised by a given network or not.

5 Computability of Weightless Neural Networks

The recognition methods used to date with weightless neural networks make them behave like finite state machines. By the other hand many neural problems, for instance, natural language understanding, make use of context free grammars, therefore it is important to be able to implement in neural networks at least some of the context-free grammars.

As mentioned before weightless networks composed of the nodes defined in section 3 and using the algorithm in section 4 are similar to probabilistic automata. In order to show that the computational power of a MPLN network and a probabilistic automaton is the same it is necessary to prove the following theorems:

- (Theorem 1) Let G_w be a weighted regular grammar and $L(G_w)$ be the language generated by G_w associated with some cut-point λ then there exists a weightless neural network, which associated with the same cut-point λ, that recognises $L(G_w)$, and
- (Theorem 2) If a set of patterns L is recognised by a weightless neural network associated with some cut-point λ then this set can be generated by a weighted regular grammar G_w associated with the same cut-point λ.

A formal proof of these theorems can be found in ([20]). Here a short description of the proofs are given.

The proof of theorem 1 is an algorithm for transforming any weighted regular grammar into a weightless network. The way in which the grammar is transformed into the neural network is such that all the properties of the grammar are preserved and it is possible to infer the grammar from the weightless network generated. Networks constructed with four kinds of nodes: *p-and*, *p-or*, *complement* and *delay* nodes will be considered. The proof is based on the complexity of the production rules. Transformation of the production rules, in weightless networks, will be started by the simplest production rule. The proof is then divided in three cases. The first case deals with production rules of the form $S_1 \rightarrow w(p)$ where w is a word in the language, S_1 is a non-terminal symbol of the grammar and p is the probability associated with this production rule. The second case deals with production rules of the form $S_i \rightarrow wS_j(p)$ and the last case deals with production rules of the form $S_i \rightarrow w_1S_j(p_1) \mid S_i \rightarrow w_2S_k(p_2)$ where $\mid$ denotes the possibility of S_i being replaced by $w_1S_j(p_1)$ or by $w_2S_k(p_2)$. Each ones of these cases are divided into sub-cases for the simplicity of the proof. For every sub-case it is given the circuit which implement it. The circuitous are designed in function of the four types of nodes mentioned above.

Although the method described to transform the weighted regular grammar into network gives a complete structure - the network and its memory contents - the language recognised by the network can be changed either by training or by changing the value of the threshold. This is particularly useful when the exact grammar of the language to be recognised is not know, only an approximation is known. The training of the network can involve only changes in the probabilities stored in the nodes or changes in any memory

position of the node. If changes are allowed in any memory position of the nodes, a different structure will be generated after the training. The generation of the network by the method described in this theorem can also be useful as an initial set up of the network.

The main purpose of the theorem 2 is to show that the relationship between weighted regular languages and weightless networks is an *if then if* relation. This proof can be used to determine the total generalisation of a network after the network has been trained. This method of calculating the total generalisation is more efficient than many others, for example: submitting patterns to the network to see if they are recognised by the net. It is also better than going through the whole network in order to calculate the generalisation. The method derived from the theorem gives also the probability of recognition for each pattern in the class recognised by the network.

Examples of the transformation of the weighted regular grammars into weightless networks can be found in ([15]) and ([19]) while an example of the transformation of a weightless network into a weighted regular grammar can be found in ([15]).

Our method of generating networks for parsing has many advantages over others models such as the method in ([14]) and in ([10]). In the model in ([14]) only finite regular grammars without recursive productions rules can be transformed in networks while ([10]) can deal with some context-free languages with the use of a stack, as an auxiliary memory. The use of stacks for pattern recognition is not very natural. It is not likely that human beings recognise patterns using stacks.

6 Turing Machine Simulation by Weightless Neural Networks

In this section we extend the WNN further (slightly and naturally) in order to simulate Turing machines ([6]). A Turing machine is a combination of a finite state machine with a device of infinite memory or cells (tape) on which it can be printed and read symbols from a finite set. The tape is scanned in either left or right direction one cell at a time. A WNN, and any Artificial Neural Network (ANN), is a finite state automata. The only missing component is the access to a tape in a similar fashion to Turing machines. This

can be achieved by encoding instructions from the output of a neural network with feedback and imagine that the input of the network is fed from a tape. This very same tape is used also to store the output of the net. We call this extension a *Turing weightless neural network*. This extension is not at all unrealistic as it may at first seem. A macro view of the human neural system as a neural black-box makes the world outside play the role of the Turing machine's tape. If we are, for example, making pen-and-paper calculations, the paper plays both the role of the input and output device.

It is important to note that the "brain" of a Turing Machine is its control: a finite state automaton. The access to a memory device (tape) is just for helping the computation: storing partial results. What makes a Turing machine computationally powerful enough as to model effective computations (Church-Turing's Thesis) is the set of permissible operations on the tape. The various classical kinds of finite automata in the computer science literature can then be roughly classified according to the set of permissible operations on their tape(s). A finite state automaton (regular languages recognisers) have one input and one output tape, but it is allowed to just move the heads to the right and as a consequence is not able to reuse past computations. A pushdown automata (context-free languages recognisers) has its access to the tape restricted to *stack* operations. A linear bounded automata (context sensitive languages recognisers) is a Turing machine with the restrictions that the head can not leave those cells on which the input was placed. While a Turing machine have unrestricted access to the tape. It is worth point out that in the four cases the tapes are assumed to be infinite and the "brain" of each type of machine is indeed a finite state automaton. In some cases it is important to distinguish between deterministic and non deterministic machines but this is outside the scope of the present work and the interested reader should consult the literature (e.g. the classic [12]).

Our point is that the fact that this memory is *potentially* infinite **should not** be considered as an intrinsical feature of a Turing machine as has been indicated in some neural computing literature. The memory is there for use but at any time the Turing machine is not allowed to use the whole infinite resource. Only finite portions of the tape can be used at any stage of a computation. It is the finitistic, repetitive and

uninspired execution of *instructions* that matters. And, as we saw above, the set of permissible operations is of fundamental significance. And these aspects of a Turing machine capture reasonably well the notion of *effective computability*. The access to an infinite tape has also obvious technical advantages. We can talk, e.g., of computations of number theoretic functions, $f : N \rightarrow N$, where N is the set of natural numbers, even if it is not possible for any existing device or any human being to actually compute the whole of a such function. Take as an example the addition of natural numbers; we all know how to perform the addition of two given numbers, but it would not be possible for us to actually compute the function $+ : N^2 \rightarrow N$ for its whole domain of definition: just imagine sufficiently large numbers whose addition would take us, say, a billion of years to perform. But that is not what matters for the study of effective computations but rather the fact that there is something mechanical, repetitive, effective, *algorithmic* about addition that we can capture. And once we have *learned* how to add a finite set of pairs of *small* natural numbers we say that we can (in principle) add any two given natural numbers when we actually know how to perform an algorithm for addition.

This idea of supplying an artificial neural networks with a head and a tape has been put forward before by McCulloch and Pitts in their seminal paper [21] for the weighted systems. This approach has been criticized as not being truly a neural network simulation [9]. We strongly disagree with that and base our simulation exactly on the idea of adding a tape to a weightless neural network.

Below we formalise the intuitive notion of a Turing machine depicted above. Amongst the various equivalent ways of defining a Turing machine we base ours upon the notation of [24][Chapter 2,pp. 40–97]. Although we do not entirely agree with the conclusions of the book, it is nevertheless an excellent *non-specialist* source for the ideas of effective computability. And more important for us, technically, is the use of a binary system of coding.

Definition 3 *A Turing Machine M over $\Sigma = \{0, 1\}$* consists of:

1. a finite set of states $Q = \{q_0, \ldots, q_n\}$ with a distinguished (*initial*) state, q_0.

2. a function $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, S\}$. δ is to be thought as a finite set of instructions and

$$\delta(q_i, \sigma) = (q_j, \sigma', m)$$

means that the machine being in the state $q_i \in Q$ and reading the symbol $\sigma \in \Sigma$ from the current cell in the tape will take the following actions:

- erase σ from the cell and write σ' in its place;
- change the internal state from q_i to q_j and
- move the head position one cell to the left ($m = L$), one to the right ($m = R$) or stops ($m = S$)

Remark 1 Observe that by asking the set of instructions to be a function, our definition requires that the machine is deterministic and must be completely specified. This is not much of a (theoretical) restriction but simplifies our proof.

The simulation can easily be grasped by observing that computations in a Turing machine is controlled by three finite state automata: one controls the head movements, the second controls the input/output and a third the next state. These automata correspond to the three components in the range of the instruction function δ in Definition 3.

We can thus easily see that we can simulate the behaviour of a Turing machine with a single layer weightless neural networks with feedback as in Figure 2. Every neuron is a k-RAM with $k = \lceil \log_2 n \rceil + 1$. The j^{th} bit of the current state vector q_i is linked (but not shown in the figure) to the input terminal x_j, $1 \leq j \leq k-1$, of every neuron. Equally, the current symbol σ is linked (but not shown) to the input terminal x_k of every neuron. The first two RAM neurons (from top to bottom) output the movements with, e.g., $L = 00$, $R = 01$ and $S = 10$; the third RAM neuron outputs symbols in Σ[2]. The remaining $\lceil \log_2 n \rceil$ neurons output the next state of the Turing machine being simulated which is then fed back to the network. The configuration in Figure 2 represents the equation: $\delta(q_i, \sigma) = (q_j, \sigma', m)$. We have thus sketchily proved the following:

[2]It is not difficult to see that if we were to allow symbols other than 0 and 1, we would have just to add $\lceil \log_2 n \rceil$ neurons and code the symbols accordingly.

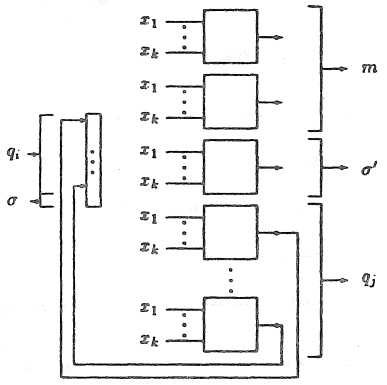

Figure 2: A Turing weightless network simulating the behaviour of a Turing machine.

Theorem 1 *Any Turing machine over* $\Sigma = \{0, 1\}$ *with* m *states can be simulated by a Turing weightless neural network with* $\lceil \log_2 n \rceil + 3$ *with* $(\lceil \log_2 n \rceil + 1)$*-RAMs neurons.*

7 Discussions and Conclusions

A study of the relationship between probabilistic automata and weightless networks was made. The main ideas behind this methodology are: (1) the proposition of a new weightless node; (2) the proposition of a new recognition algorithm; (3) the proof of the equivalence between probabilistic automata and weightless networks and (4) the increasing of the computation power of weightless networks.

The main strengths of the method are that: (1) the ability to construct weightless networks to recognise any weighted regular language; (2) the possibility of parallel implementations in hardware for compilers based on weighted regular languages and (3) to provide a deterministic decision whether a pattern belongs to a language or not.

Regular grammars can be transformed in RAM nets and vice versa using the same algorithms of the proofs of theorems 1 and 2; but where the neuron stores 0, 1 and not probabilities. Regular grammars and RAM

neurons are special cases of weighted regular grammars and weightless neurons respectively, as a consequence it is clear that the algorithms will work for them. It is well known that RAM nets are finite state machines and so they are able to recognise finite state languages. No weighted regular grammar of languages other than regular ones can be implemented in RAM nets, since RAM nets are finite state machines and finite state machines can only recognise regular languages. If MPLN nodes, with the conventional recognition algorithm, are used instead of RAM, the functions which can be recognised will be the same ones. The difference between RAM and MPLN networks is the same as the difference between deterministic and non-deterministic automata. It was shown that networks constructed with the nodes and algorithms in this paper can compute all weighted regular languages and RAM and MPLN networks can only compute regular languages then we can conclude that the model here is more powerful than the previous ones.

There are several interesting points that were not examined in this paper as for instance the practical applications of weighted regular languages. There are weighted regular languages which are context-free, context-sensitive and even recursive languages. The possibility of dealing with weighted regular languages by itself is, then, a very good result but the practical shortcomings of such a result was not really investigated. Clearly, this investigation needs to done soon.

The Turing Machine simulation is extraordinaryly simple if one compares with similar equivalence proofs using the weighted model [28]. [9]. In their approach the infinite tape is simulated "inside" the net via the weights of the connections or by assuming infinite amount of nodes. The former requires unbounded weights and both requires very intricate encoding. It is worth pointing out that, as our model is weightless, there is no conceivable way to employ the techniques used in [29, 28] where the tape is represented in the weights of the connections. The weighted approaches differ among themselves by using

- infinite number of nodes: [9], [11], [29];
- finite number of high order nodes with unbounded weights: [25];
- finite number of nodes with linear-interconnections but unbounded weights: [28]. It is worth mentioning that this is achieved by using rational weights.

We could have used an *internal* representation of the Turing machine's tape by allowing an infinite number of neurons, as in [9, 29], but we find that very unnatural.

Our results establish, at a theoretical level, the relationship between conventional and neural models of computation and allow for the (theoretical) possibilities of using ANN in "high-level" cognitive tasks such as natural language processing and weightless or common sense reasoning, that have traditionally been dealt with the symbolic representation and processing techniques of Artificial Intelligence.

In relation to the controversy surrounding the distinction between symbolical and neural methods of computation, we have the position that the two are variants of the notion of effective computability. The results here give further evidence to that. Our earlier research [20], [18], [19] can be seen as an evidence that neural computing is not inappropriate neither incompatible as a way of modeling cognitive tasks [8] and we claim that we have given further evidence of this with the results showed in this paper.

The problem of learning effective computable functions is being investigated in [7].

References

[1] R. Al-Alawi and J. Stonham. A training strategy and functionality analysis of multilayer boolean neural networks. Technical report, Brunel University, 1989.

[2] I. Aleksander. Emergent intelligent properties of progressivelystructured pattern recognition nets. *Pattern Recognition Letters*, 81:375–384, 1983.

[3] I. Aleksander. Adaptive pattern recognition systems and boltzmann machines: A rapprochement. *Pattern Recognition Letters*, 1987.

[4] I. Aleksander. The logic of connectionist systems. In R. Eckmiller and C. Malsburg, editors, *Neural Computers*, pages 189–197. Springer-Verlag, Berlin, 1988.

[5] I. Aleksander and H. Morton. *An Introduction to Neural Computing*. Chapman and Hall, London, 1990.

[6] W. R. de Oliveira and T. B. Ludermir. Turing machine simulation by logical neural networks. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks II*, pages 663-668. North-Holland, 1992.

[7] W. R. de Oliveira and T. B. Ludermir. Learnability of logical neural networks. Submitted for Publication, 1993.

[8] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: a critical analysis. *Cognition*, 28:2 71, 1988.

[9] S. Franklin and M. Garzon. Neural computability. *Progress in Neural Networks*, 1:128,144, 1990. Ablex, Norwood.

[10] C. Giles, G. Chen, H. Chen, and Y. Chen. Higher order recurrent networks and gramatical inference. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, 1990.

[11] R. Hartley and H. Szu. A comparison of the computational power of neural network models. In *Proc. IEEE Conf. Neural Networks III*, pages 17,22, 1987.

[12] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[13] W. K. Kan and I. Aleksander. *Neural Computing Architectures*, chapter RAM-Neurosn for adaptive image transformation tasks. Chapman and Hall, 89.

[14] S. Lucas and R. Damper. A new learning paradigm for neural networks. In *Proceedings First IEE International Conference on Artificial Neural Networks*, 1989.

[15] T. B. Ludermir. *Automata theoretic aspects of temporal behavior and computability in Logical neural networks*. PhD thesis, Imperial College, London, UK, December 1990.

[16] T. B. Ludermir. A feedback network for temporal pattern recognition. In R. Eckmiller, editor, *Parallel Processing in Neural Systems and Computers*, pages 395–98. North-Holland, 1990.

[17] T. B. Ludermir. A cut-point recognition algorithm using PLN node. In *Proceedings of ICANN-91*, Helsinki, 1991.

[18] T. B. Ludermir. Logical networks capable of computing weighted regular languages. In *Proceedings of IJCNN-91*, Singapore, 1991.

[19] T. B. Ludermir. Logical neural nets and distributed implementations of weighted regular languages. In *International Conference on Artificial Neural Networks*, Proceedings of the IEE 349, Bournemouth, 1991.

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

[21] W. McCulloch and W. Pitts. A logical calulus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[22] C. E. Myers. Learning algorithms for probabilistic neural nets. In *Proceedings First INNS Annual Meeting*, 1988.

[23] C. E. Myers. Output functions for probabilistic logic nodes. In *Proceedings First IEE International Conference on Artificial Neural Networks*, 1989.

[24] R. Penrose. *The Emperor's New Mind: concerning computers, minds and the laws of Physics*. Oxford University Press, vintage edition edition, 1990.

[25] J. Pollack. *On connectionist models of natural language processing*. PhD thesis, Computer Science Dept, Univ. of Illinois, 1987.

[26] M. O. Rabin. Probabilistic automata. *Information and Control*, 6:320–345, 1963.

[27] A. Salomaa. Probabilistics and weighted grammars. *Information and Control*, 15:529–44, 1969.

[28] H. Siegelman and E. D. Sontag. Neural nets are universal computing devices. Technical Report SYCON-91-08, Rutgers Center for Systems and Control, 1991.

[29] G. Sun, H. Chen, Y. Lee, and C. Giles. Turing equivalence of neural networks with second order connection weights. In *Proc. Int. Joint Conf. Neural Networks*, 1991.

[30] J. G. Taylor. Noisy neural net states and their time evolution. Technical report, Kings College London, 1987.