A Simple Architecture of High Order Network with Positive Bounded Integer Weights

by

Adam Kowalczyk, Greg Aumann and Jacob Cybulski

Abstract

In any practical implementation of neural networks, weights have to be restricted to a finite set of values. In current hardware implementations the number of such available values have to be very small indeed, so they have to be taken explicitly into account at the design stage.

The paper presents and discusses some basic properties of modulo perceptrons which are neural networks that explicitly take the limited size of possible weights into account. These networks are a form of polynomial discriminators or high order, feed forward networks with positive integer weights of limited magnitude. We illustrate the concept on a simple example, then we outline results concerning generation of such networks, in particular some conversion theorems from the real to modulo weight case, and finally show that the structure allows very simple digital hardware implementation.

A Simple Architecture of High Order Network with Positive Bounded Integer Weights

by

Adam Kowalczyk and Greg Aumann
Telecom Australia
Research Laboratories
770 Blackburn Road, Clayton Vic 3168

Jacob L. Cybulski
Amdahl Australian Intelligent Tools Programme
Dept of Comp. Sci. and Comp. Eng., La Trobe Uni
Bundoora Vic 3083

Introduction

In any practical implementation of neural network, weights have to be restricted to a finite set of values. Although this limitation is hardly noticeable in digital simulation, it becomes a critical issue when it comes to dedicated hardware implementations, be they digital or analog [5]: the number of possible weight values has direct impact on the size of neural network implemented on a single chip.

In this paper we discuss a class of modulo perceptrons [1,2] which are neural networks that explicitly take the limited size of possible weights into account. The main innovation is in introduction of an extra step taking modulo of the final sum before imposing thresholds or performing some other processing. This simple trick allows, among other things, unsigned digits of limited size as weights of links, and so eliminates the need for negative weights, which are a nuisance in the case of analog hardware implementations.

In this paper we will introduce some background relating modulo perceptrons to some other concepts, before introducing them formally. We will then outline some results on the generation of such networks and discuss a simple example. Subsequently digital hardware architecture will be presented followed by a brief discussion and conclusions.

Background

From the early days of artificial neural networks research, the potential advantages of high order networks (polynomial classifiers) for dichotomization of pattern space were recognized [11]. In the 1950's and 1960's however, the high order units were not used practically, mainly due to lack of efficient techniques to cope with the combinatorial explosion in the number of high order terms. In the current wave of neural net activities, a number of researchers have turned attention to high order networks again, motivated by a variety of reasons including increase in memory capacity[13], capability of high order terms to embed prior knowledge about properties of a domain of interest [4], optical implementation of associative memory[12], universal capability of structure to implement any predicate [10] and recent availability of very efficient training procedures based on empirical selection or even generation of useful terms [1,2,6,7]. The concepts of modulo perceptron and its particular case binary perceptron, which are natural adaptations of single slab high order network to the case of a limited set of available weights, was studied in some of our earlier papers [1,2,7].

Modulo perceptron

A *high order unit* is a combination $y = S \circ P(x)$ of a function $S: R \rightarrow R$ with a polynomial $P: R^n \longrightarrow R$. Three particular cases of high order units defined on a set $X \subset \{0,1\}^n$ of binary vectors are of special interest to us (we consider a single output case here, for simplicity):

(i) A higher order unit with the step function non-linearity, $S := \theta(t)$ defined as 1 for $t \geq 0.5$, and 0 otherwise, will be called a *mask-perceptron*. We can write in this case:

$$y = f(x) := \theta\Big(\sum_{i \in I} w_i x^i\Big) \qquad (x \in \{0,1\}^n), \tag{1}$$

where I is a set of n-tuples $i=(i_1, \ldots, i_n) \in \{0,1\}^n$, $x^i := x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$ and $w_i \in R$.

(ii) A *mod(M)-perceptron* ($M \in \{2, 3, \ldots\}$), or generically a *modulo-perceptron*, is defined as:

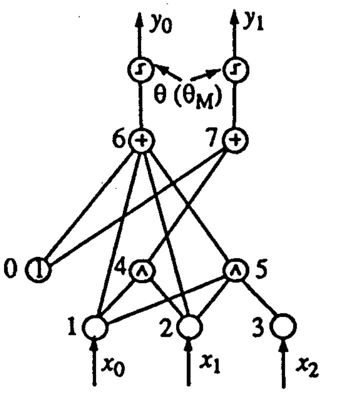
$$y = f_M(x) := \theta_M\Big(\sum_{i \in I} u_i x^i\Big) \qquad (x \in \{0,1\}^n), \tag{2}$$

where $u_i \in \{0, 1, \ldots, M-1\}$ and $\theta_M(t) := 1$ if $0.25 \times M \leq t \bmod M \leq 0.75 \times M$ and 0 otherwise; modulo-perceptrons for $M = 2^n$ are of special interest in this paper.

(a) Decision Table

input			output	
x_0	x_1	x_2	y_0	y_1
1	1	1	1	0
1	0	1	0	1
0	1	1	0	1
0	0	1	1	1

(c) Architecture



(c) Weights to the output links

Nodes	Type of perceptron					
	mask		*mod*(8)		binary	
to / from	6	7	6	7	6	7
0	1.20	0.80	5	3	1	1
1	-0.90		4		1	
2	-1.10	0.10	4	0	1	0
4	2.13	-0.88	1	4	1	1
5	-0.33		7		1	

Example of three high order networks with the same architecture implementing the same decision table. The *mod*(8) perceptron was obtained by rounding weights of the mask perceptron in 0.25 precision, and then applying conversion theorem (ii). A number of large modulo perceptrons generated in this way was reported in [1,7]. The binary perceptron was generated differently by direct selection of terms; paper [2] reports a number of large binary perceptrons generated similarly.

Figure 1

(iii) A *binary-perceptron* which is by definition *mod(2)*-perceptron. In this case $u_i \in \{0, 1\}$, and θ_2 becomes a parity function, equal 0 if t is even and 1 if it is odd.

The main advantage of mask perceptrons is their simplicity: masks x^i are logical conjunctions [10], so evaluation of $\sum_{i \in I} w_i x^i$ is extremely easy to implement in hardware since it involves no multiplication. The implementation of modulo perceptrons and binary perceptrons, in particular, is even easier: weights are unsigned integers of pre-determined maximal size or single bits, respectively. It is also worthwhile to add that there exist efficient algorithms for empirical designs of such networks [6].

The following results give a sample of relations between mask perceptrons (Eq. 1) and modulo perceptrons (Eq.2).

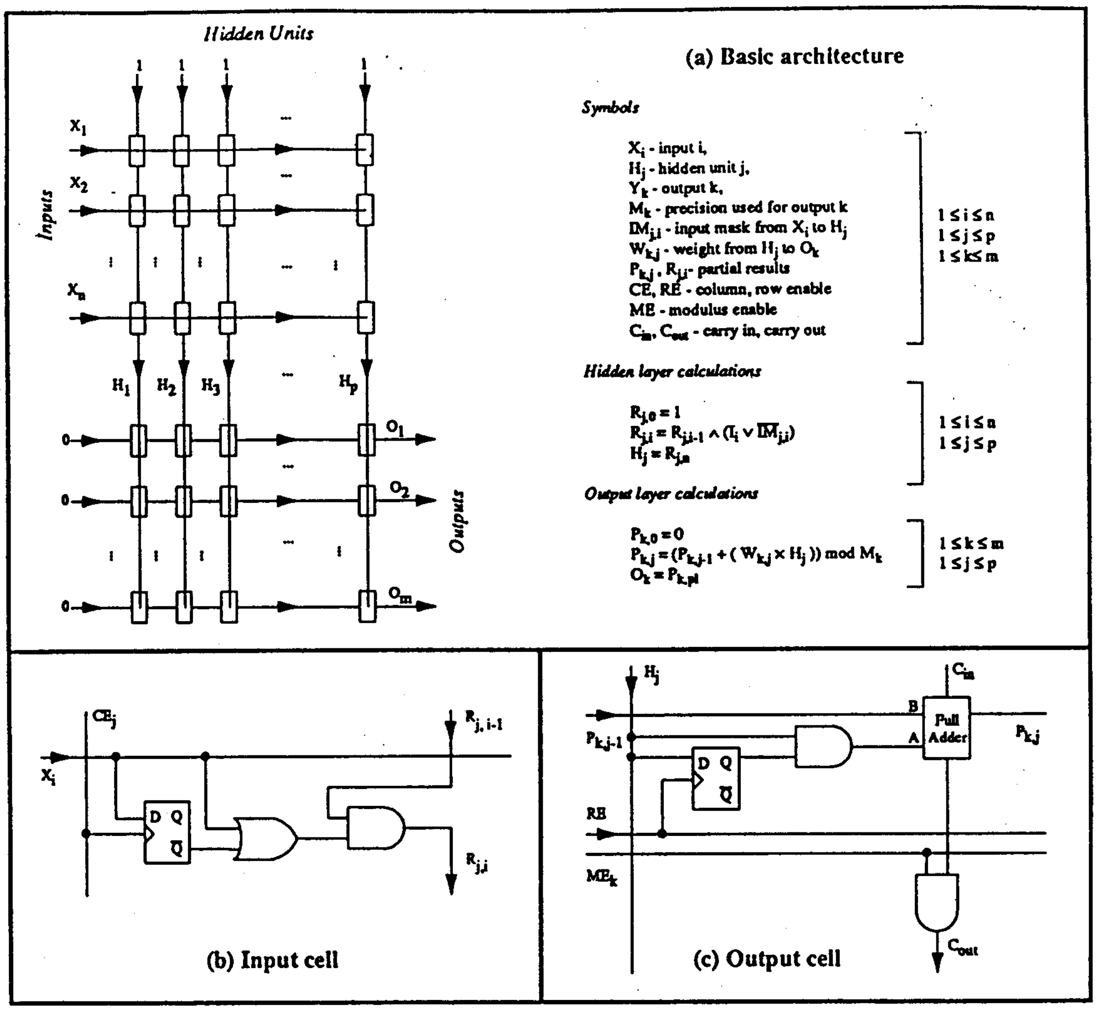
(i) *Universality*. Any dichotomy of $X \rightarrow \{0, 1\}$ can be implemented as a mod(M)-perceptron with prime M (a binary-perceptron in particular).

(ii) *Conversion theorem*. If the weights w_i in (1) are rational numbers, i.e. $w_i = k_i / M$ with integers k_i, M (M>0), then the mask-perceptron $f(x)$ defined by (1) and mod($2M$)-perceptron $f_{2M}(x)$ given by (2) with weigths $u_i := k_i \bmod (2M)$, provide identical results for every x such that $-0.5 < \sum_{i \in I} w_i x^i \le 1.5$.

(iii) *Modulo prime expansion*. Let M be a prime number (M = 2 in particular), and a, c be real numbers such that $a + (M-1)c/2 < 0.5 \le a + (M+1)c/2$ and $\sum_{i \in I} w_i x^i \in \{a, a+c, \ldots, a+(M-1)c\}$ for every $x \in X$. Then there exists a a mod(M)-perceptron (2) providing identical outputs as the mask-perceptron (1) for every $x \in X$.

An Implementation Architecture

The two sets of basic equations in Fig. 2 define two basic building blocks which can be used to implement a modulo perceptron. The implementation can be serial in which case it has a time complexity of O(np+pm). However it can also be easily implemented in parallel. This could be on a parallel computer such as the connection machine or direct-



Symbols

X_i - input i,
H_j - hidden unit j,
Y_k - output k,
M_k - precision used for output k
$IM_{j,i}$ - input mask from X_i to H_j
$W_{k,j}$ - weight from H_j to O_k
$P_{k,j}$, $R_{j,i}$ - partial results
CE, RE - column, row enable
ME - modulus enable
C_{in}, C_{out} - carry in, carry out

$1 \leq i \leq n$
$1 \leq j \leq p$
$1 \leq k \leq m$

Hidden layer calculations

$R_{j,0} = 1$
$R_{j,i} = R_{j,i-1} \wedge (I_i \vee \overline{IM}_{j,i})$
$H_j = R_{j,n}$

$1 \leq i \leq n$
$1 \leq j \leq p$

Output layer calculations

$P_{k,0} = 0$
$P_{k,j} = (P_{k,j-1} + (W_{k,j} \times H_j)) \bmod M_k$
$O_k = P_{k,p}$

$1 \leq k \leq m$
$1 \leq j \leq p$

Figure 2

ly in a VLSI chip. A VLSI implementation is particularly simple and is the one discussed in this paper. The discussion initially assumes that the weights will be stored in memory on the chip. This allows the network to be quickly retrained. Later in the discussion some alternative ways of storing the weights and the benefits of these approaches will be examined.

The architecture of a binary perceptron chip can consist of two rectangular blocks which are matrices of two types of simple cells. The first one takes the inputs and calculates the values of the hidden units. The second block takes the values of the hidden units and calculates the values of the outputs. This architecture is shown in Fig. 2a. The simple cells each implement one pass of the two basic equations shown above. The hidden layer calculations shown in Fig. 2 are obviously simple to implement as each of the terms and the result are binary values. The output layer calculations are only a little more complex. This is because again all the values are binary and so the multiplication reduces to an AND operation. The modulus operation is also simple because it just takes the least significant bit (in the binary case).

This architecture can be easily extended for mod(2^m)-perceptrons. If this is the case then we can group output bits together to form a single multiple bit output. To do this we need to add the facility to selectively pass carry bits across output bits. This is easily done with the addition of an AND gate.

The other requirement is a means to load the weights into the network. This can be done by adding column enables on the input block and using the inputs to carry the weights. For the output block it can be done by adding row enables, breaking the hidden unit lines in between the two blocks and using them to carry the weights. An extra line also needs to be added to the output units to indicate whether to pass the carry bit to the next output or not. This would be 0 for the most significant bit of each output and 1 otherwise. Thus for binary outputs they would all be 0. Diagrams of the two basic types of cells which store weights in on chip memory are shown in Fig. 2.

These cells can be simplified considerably if less flexibility in reconfiguring the network is required. Two alternatives are that the weights could be stored in EEPROM cells or they could be mask programmed at manufacture. EEPROM cells allow the network to be reprogrammed but it would require removal of the chip and approximately an hour. The use of EEPROM storage of weights should allow smaller cells and so more cells per chip. Mask programming doesn't allow any changes to be made but conversely allows many more units on a chip. Using mask programming allows the simplification of the input cells to a single transistor which will only be present if the input weight is 1. This is in a structure very similar to that of the AND plane in a PLA (Programmable Logic Array [8]). It is however simpler because the complement of the input is not needed. The output cells cannot be simplified as much as the input cells. They will still need to contain an adder but the flip flop and the AND gates will be eliminated. It may seem that a mask programmed network is not flexible enough to be useful in practice. This is not the case. If neural networks are to be used in consumer equipment then most users will not be sophisticated enough to train the networks. Thus the networks must come pre-trained or some form of training that is not visible to the user must be used.

Conclusions

In some situations, a single slab higher order network can be converted to a modulo perceptron (mod(M)-perceptron) with bounded, positive integer values of link weights.

Mod(2^m)-perceptrons are especially easy to implement in digital VLSI with currently available technologies.

Algorithms dedicated explicitly to generation of modulo perceptron should be investigated.

Acknowledgement

We would like to express our thanks to Mr. Chris Rowles for its helpful comments during preparation of this paper. The permission of Executive General Manager of Telecom Australia Research Laboratories to publish this paper is also acknowledged.

References

[1] J.L. Cybulski, H.L. Ferrá, A. Kowalczyk, and J. Szymański, Experiments with multi-layer perceptrons, in *Proc. of the Australian Joint Artificial Intell. Conf.* (AI'89, Melbourne 1989).

[2] J.L. Cybulski, H.L. Ferrá, A. Kowalczyk, and J. Szymański, Determining Word Lexical Categories with a Multi-Layer Binary Perceptron, in *Expert Systems and Neural Networks: Theory and Applications, Proc. Fifth IASTED Int. Symp.*, ed. Hamza, M.H., ACTA Press 1989.

[3] L. Cybulski, A. Kowalczyk, A. Jennings, and F. Morselt,The Implementation of Two-Layer Binary Perceptrons, to appear in *Proc. of the Australian Joint Artificial Intell. Conf.* (AI'90, Perth 1990).

[4] C.L. Giles, R.D. Griffin, and T. Maxwel, Encoding geometric invariances in higher-order neural networks, Proc. Neur. Inf. Proc. Sys., Dever 1987, Amer. Ins. Phys., NY, 1988, pp. 301-9.

[5] P. H. Graf, and L. D. Jackel, Analog Electronic neural Network Circuits, IEEE Circuits and Devices Magazine, pp.44-55, July 1989.

[6] A. Kowalczyk, Optimization of Decision Operator, CSS Branch Paper 179, Telecom Australia, 1989.

[7] A. Kowalczyk and H.Ferra, Experiments in Lexical Classification by Multi-Layer Perceptrons with Simplified Interconnection Weights, to appear in *Proc. of the Australian Joint Artificial Intell. Conf.* (AI'90, Perth 1990).

[8] J. Millman, *Microelectronics: Digital and Analog Circuits and Systems*, McGraw Hill, 1979.

[9] T. Maxwell, C. L. Giles, Y. C. Lee, and H. H. Chen, Nonlinear Dynamics of Artificial Neural Systems, in J. Decker (Ed.) AIP Conf. Proc., NY: Amer. Inst. of Phys., 1986

[10] M. Minsky, and S. Papert, *Perceptrons*, MIT Press, Cambridge, Massachusetts, 1969 (edition 1988).

[11] N. Nilson, Learning Machines, McGraw-Hill, NY,1965.

[12] D. Psaltis, C. H. Park, and J. Hong, Higher Order Associative Memories and Their Optical Implementations, Neural Networks, Vol.1 (1988), pp. 149-163.

[13] D. Psaltis and C.H. Park, Nonlinear discriminant function and associative memory, in J. Decker (Ed.) AIP Conf. Proc., NY: Amer. Inst. of Phys., 1986, pp. 370-75.