Neural Processing Letters **12:** 247–254, 2000.

On the Internal Representations of Product Units

JUNG-HUA WANG, YI-WEI YU and JIA-HORNG TSAI
Department of Electrical Engineering, National Taiwan Ocean University, 2 Peining Rd., Keelung, Taiwan. e-mail: b0141@ind.ntou.edu.tw

Abstract. This paper explores internal representation power of product units [1] that act as the functional nodes in the hidden layer of a multi-layer feedforward network. Interesting properties from using binary input provide an insight into the superior computational power of the product unit. Using binary computation problems of symmetry and parity as illustrative examples, we show that learning arbitrary complex internal representations is more achievable with product units than with traditional summing units.

Key words: product unit, internal representations, recurrent neural networks, perceptrons, backpropagation training.

1. Introduction

Internal representation normally refers to the pattern of activation on the hidden nodes that correspond to the encoding of the input pattern; or some literature [6] may simply refer to it as the set of connection weights between input and hidden units required to perform a specific input pattern encoding. Analyzing internal representation of hidden nodes can be an effective approach in characterizing the computational properties of the underlying network. For example, it has been shown that the second order recurrent network [2], which is based on Sigma-pi units to learn finite state automata, attempts to form clusters in activation space as its internal representation of states, causing the state machine to deteriorate for long input strings. For this problem, Zeng et al. [3] proposed a pseudo-gradient learning rule that uses isolated point in activation space instead of vague clusters as its internal representation of states to force a second order recurrent network to learn stable states. These works showed that the second order networks learn finite state machine more easily than the first order simple recurrent network structure [4], because the former has greater internal representational power than the latter.

The present paper focuses on analyzing the internal representational power of product unit when it is used as a hidden-layer node in a multi-layer feedforward network, and demonstrating its use in solving hard binary computation problems, e.g., the XOR problem and the symmetry problems. The intention behind these demonstrations is threefold. Firstly, it provides insight into the computation nature of the product unit. Secondly, in the field of digital logic design the basic function such as XOR is taken as a *universal* gate from which any arbitrary logic circuits

can be built. Finally, problems of XOR, symmetry, and parity are well known and widely used as benchmarks in literature [5] to demonstrate various neural networks learning capability. Unlike [2] and [3], the goal here is not to construct a higher-order recurrent network (although theoretically it can be done with product units) to learn complex tasks such as finite state automata or language grammars. Instead, we focus strictly on the component level, i.e., product unit employed as a hidden node in a multi-layer feedforward network. Thus the main aim of this paper is to obtain better understanding of the product unit by analyzing its computation nature in the context of internal representation, to verify our analytical results by demonstrating how some difficult binary problems can be easily solved by using product units, which in turn may give us more insight into their capability for fulfilling other more complicated tasks.

2. Mathematical Analysis

In contrast to the linear combinatorial terms in the summing units (e.g. perceptrons [7]) and polynomial terms in sigma-pi units (e.g. second order networks take the form $y_i = F(\sum w_{ijk} x_j s_k)$), product units allow fractional and even negative terms. Therefore, it is expected that more powerful internal representation capability might be obtained by using product units. The output of a product unit is given by

$$y = F(g - \theta_h), \quad g = \prod_i^N x_i^{w_i} \tag{1}$$

where $F(\,)$ is called the activation function which normally takes the form of a sigmoidal or threshold function, g is the total input stimulus, $\theta_h =$ threshold, and w_i is the connection weight between input x_i and the product unit. As w_i is on the exponential term, learning arbitrary complex internal representation can be achieved by adjusting w_i. We start by rewriting Equation (1) as

$$g = e^{\sum_i^N w \log|x_i|} \left(\cos \pi \sum_{x_i<0} w_i + i \sin \pi \sum_{x_i<0} w_i \right)$$

If x_i is real,

$$g = e^{\sum_i^N w_i \log|x_i|} \left(\cos \pi \sum_i^N w_i I_i \right),$$

where $I_i = 1$ if $x_i < 0$, and $I_i = 0$ otherwise.

Now, Equation (1) can be simply written as

$$g = e^u \cos \pi V, \quad \text{where} \quad u = \sum_i^N w_i \log|x_i|, \tag{2}$$

$$V = \sum_{i}^{N} w_i I_i \tag{3}$$

For binary inputs ($x_i = 1$ or $x_i = -1$), as $e^u = 1$ Equation (2) can be further simplified to

$$g = \cos \pi V. \tag{4}$$

In the following discussions, the above results will be applied to solving hard binary problems such as the symmetry, parity, and XOR. Note that these binary problems have been often used as illustrative examples and benchmarks in literature [1,5,7] where they were solved using various neural networks such as multi-layer perceptrons or recurrent networks trained with lengthy backpropagation training algorithm [6] or its variants.

3. Solving the Symmetry Problem

The first interesting problem we study is that of classifying a binary input string as to determine whether or not it is symmetric about its center. For example, $(-1, 1, -1|-1, 1, -1)$ is a 6-dimension symmetric string, whereas $(-1, 1, 1|1, -1, 1)$ is not. Such an input string has been shown [6] solvable by backpropagation training a 3-layer perceptron network with two *summing* units in the hidden layer. However, due to the explosive number of possible input combinations, training such a network for classifying high-dimensional strings requires unrealistically lengthy computation time. In contrast, the symmetry problem can be easily solved by using a simpler product network (PN) shown in Figure 1, where the hidden layer has a single product unit (denoted by Π) and output layer has a summing unit (denoted by Σ). To explain, we first note that from Equations (2) and (3), connection weights associated with positive inputs are trivial. Thus, $V = w_1 + w_3 + w_4 + w_6$ for the input string $(-1, 1, -1|-1, 1, -1)$. Using this property and letting the activation of the hidden node y^h be determined by the following threshold function

$$y^h = F_h(\cos \pi V - \theta_h), \qquad F_h(g) = \begin{cases} 1, & \text{if} \quad g > 0 \\ 0, & \text{if} \quad g \leqslant 0 \end{cases}, \tag{5}$$

the internal representation of the network shown in Figure 1 can be readily obtained by using the following steps:

(1) Place a single product unit in the hidden layer and set $\theta_h = 0$. Then, assign a positive number, e.g. 3, to the connection weight between the output and the hidden node. The activation function F for the summing unit is also set to be a threshold function.

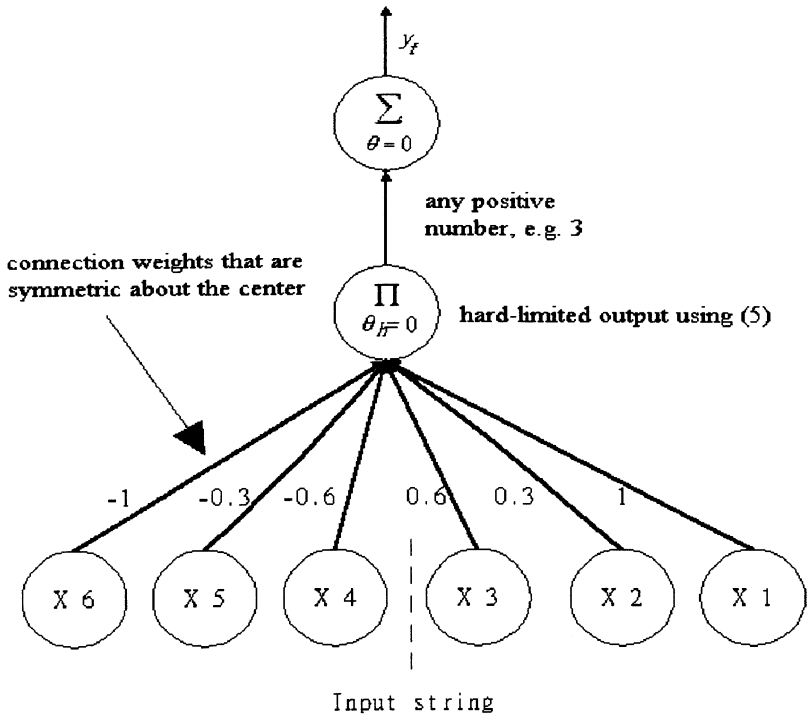

Figure 1. PN for identifying symmetric input strings.

(2) Assign the connection weights between the hidden and input units with numbers that are equal in magnitude and opposite in sign (i.e., symmetric about the center, as indicated by the dashed line in Figure 1).

Then, the input string is symmetric if the thresholded output (of the summing unit) $y_f = 1$, otherwise the string is non-symmetric. The key point to see about this algorithm is that V always equals 0 for a symmetric input string. Unlike the solution found in [6] where two hidden summing units and roughly 1208 input presentations are needed to train for correctly classifying 6-element input strings, whereas the PN requires only a hidden product unit, regardless of the input dimension. Moreover, as stated in [6], in applying summing units to solve the symmetry problem, one has to make sure that the connection weights on each side of the midpoint of the string being in the ratio of 1:2:4. In the PN, one does not have this awkward limitation. Finally, as no lengthy training is required in this example of using the product unit to solve the symmetry problem, we in a sense verified that the PN is more computationally efficient than the all-summing-unit network.

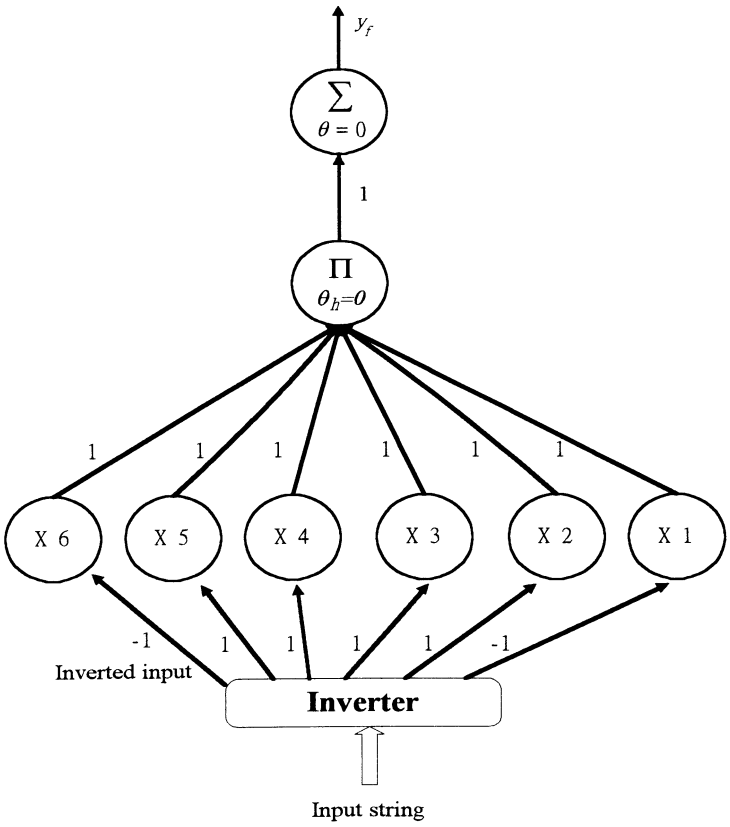

Figure 2. PN for solving the parity problem. Test input string is (1, −1, −1, −1, −1, 1).

4. Solving the Parity Problem

Classifying input parity is a very difficult problem because the most similar strings require different answers. One should note that the famous XOR problem is just a special case of parity problem with input dimension = 2. In this work, the parity problem is solved by using a PN in which the network output $y_f = 1$ if the input string contains even number of 1's, and $y_f = 0$ otherwise. To proceed, we first set all input connection weights in Figure 2 to 1. The purpose of doing so is to make the response of output unit solely depends on the total number of negative inputs. For example, given the input string (1, −1, −1, −1, −1, 1), the output of the product unit y is calculated by

$$y = \prod_i^N x_i^{W_i} = (1)^1(-1)^1(-1)^1(-1)^1(-1)^1(1)^1 = (-1)^{1+1+1+1} = (-1)^{\sum_i^N W_i I_i}$$

That is, the thresholded output $y_f = 1$ if number of negative inputs is even, and $y_f = 0$ otherwise. In essence, it is the number of positive inputs (i.e., 1) that accounts for the final output. This observation naturally leads to the use of an inverter in Figure 2. We first invert the input string x to $\overline{x}$, then the output of Equation (1) is expressed as

$$y = \prod_{i}^{N} \overline{x}_i^{w_i} = (-1)^1(1)^1(1)^1(1)^1(1)^1(-1)^1 = (-1)^{\text{total number of } -1\text{s}} = (-1)^2 = 1$$

Thus, the internal representations of the PN for solving the parity problem are determined by using the following steps

(1) Complement the input binary string.
(2) Place a single product unit in hidden layer and assign zero to thresholds of both the hidden unit and output unit.
(3) Assign constant 1 to *all* connection weights of the network.

Here, the most significant point to note is the assignment of constant 1 to all connection weights, which require no training. On the other hand, if summing units are used in the hidden layer, then N summing units are required to solve an N dimensional input string [6]. Take a *4-D* input string as an example, the number of needed input presentations can be as many as 2825 during the backpropagation training course. Clearly, training summing units to classify high dimensional input strings is not practical, if not impossible. In contrast, a *single* product unit is sufficient for solving the parity problem that has input strings with arbitrary dimensions. This example, as well as the one presented in the last section, clearly demonstrates that the product unit has greater internal representation capability than the summing unit. Finally, for comparison, we also used the backpropagation algorithm to train a PN to classify parities for lower-dimensional input strings. Figure 3 illustrates the resulting network for classifying five-element input strings.

5. Discussion and Conclusions

Using binary problems of the symmetry, parity, and XOR as illustrative examples, we have successfully shown the superior internal representation power of the product unit. Although the product unit has been shown [1, 8] more general than traditional summing unit or higher order unit [6, 11], it has received less attention than it deserves. With recently fast developing soft computing techniques [13] such as fuzzy theory and genetic algorithms, we believe that product unit is very promising for optical and electronic VLSI implementations [9, 10] in dealing with computation problems that involve more general *continuous inputs*. For future works, the product unit will be used to replace the sigma-pi unit in the recurrent network [2, 3] for

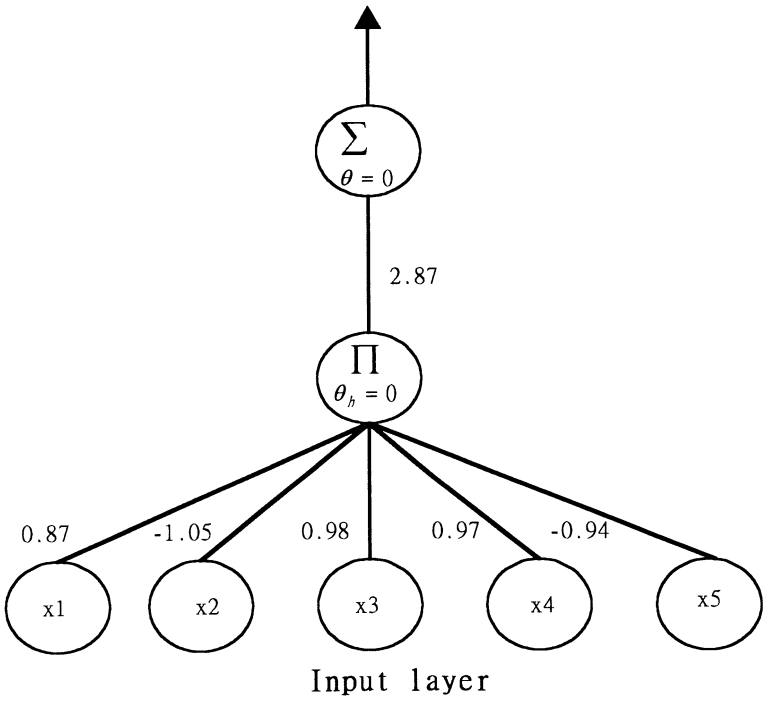

Figure 3. Resulting network after backpropagation training a 5-element odd parity input string.

learning finite state automata; the required modifications to the pseudo-gradient learning rule [3] due to using product units can make use of the derivations in [12].

Acknowledgements

This research was supported by National Science Council of Taiwan under Grant number NSC86-2611-E019-024.

References

1. Durbin and Rumelhart, D. E.: Product units: a computationally powerful and biologically plausible extension to backpropagation networks, *Neural Computation* **1** (1989), 133–142.
2. Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z. and Lee, Y. C.: Learning and extracting finite state automata with second order recurrent neural networks, *Neural Computation* **4** (3), (1992), pp. 393–405.
3. Zeng, Z., Goodman, R. M. and Smyth, P.: Learning finite state machines with self-clustering recurrent networks, *Neural Computation* **5** (1993), 976–990.

4. Elman, J. L.: Distributed representations, simple recurrent networks, and grammatical structure, *Machine Learning* **7** (1991), 195–225.
5. Lin, C. T. and Lee, C. S.: *Neural Fuzzy Systems, A Neuro-Fuzzy Synergism to Intelligent Systems*, Prentice-Hall, 1996.
6. Rumelhart, D. E. and McClelland, J. L.: *Parallel Distributed Processing, Explorations in the Microstructure of Cognition,* Vol.1, MIT Press, Cambridge, 1986.
7. Minsky, M. and Papert, S.: *Perceptrons*, MIT Press, Cambridge, 1969.
8. Chen and Bastani, F.: ANN with two-dendrite neurons and its weight initialization, *Proc. of Internat. Joint Conf. on Neural Networks*, 1992, Vol. 3, pp.139–146.
9. Chung, P. C. and Krile, T.F.: Reliability characteristics of quadratic Hebbian-type associative memories in optical and electronic network implementations, *IEEE Trans. Neural Networks* **6**(2) (1995), 357–367.
10. Wolpert, S. and Micheli-Tzanakou, E.: A Neuromime in VLSI, *IEEE Trans. Neural Networks* **7**(2) (1996), 300–306.
11. Baldi, P. and Venkatesh, S.: Random interactions in higher order neural networks, *IEEE Trans. Inf. Theory* **39** (1993), 274–283.
12. Wang, J. H. and Jeng, M. D.: Performance characterization of product unit neural network associative memories, *Proc. of 9th Internat. Conf. CAD/CAM, Robotics and Factories of the Future*, Newark, NJ, August 1993.
13. Jang, J. S. R., Sun, C. T. and Mizutani, E.: *Neuro-Fuzzy and Soft Computing*, Prentice-Hall, 1997.