Published in *Proceedings of IEEE/INNS International Joint Conference on Neural Networks*, IJCNN'93-Nagoya, Oct. 25-29, Vol.1, pp. 589-592 (1993).

(a) **The title of the article**
A Back-Propagation Algorithm for Neural Networks Based on 3D Vector Product

(b) **The authors' full names**
Tohru Nitta

(c) **Current Affiliations**
Neuroscience Research Institute,
National Institute of Advanced Industrial Science and Technology (AIST),
AIST Tsukuba Central 2, 1-1-1 Umezono, Tsukuba-shi, Ibaraki, 305-8568 Japan.
E-mail: tohru-nitta@aist.go.jp

A Back-propagation Algorithm for Neural Networks Based on 3D Vector Product

Tohru Nitta

Electrotechnical Laboratory,
1-1-4 Umezono, Tsukuba Science City, Ibaraki, 305 Japan.

Abstract :
A 3D vector version of the back-propagation algorithm is proposed for multi-layered neural networks in which vector product operation is performed, and whose weights, threshold values, input and output signals are all 3D real numbered vectors. This new algorithm can be used to learn patterns consisted of 3D vectors in a natural way. The XOR problem was used to successfully test the new formulation.

1 INTRODUCTION

We present a three-dimensional vector version of the back-propagation algorithm (called "VP-BP"(Vector Product Back Propagation)), which can be applied to multi-layered neural networks whose weights, threshold values, input and output signals are all 3D real valued vectors, and are computed using 3D vector product that is invented by the demands of sciences, e.g. dynamics. There has already existed a back-propagation learning algorithm for patterns consisted of 3D vectors [4] (3DV-BP), which can be applied to multi-layered neural networks whose threshold values, input and output signals are all 3D real valued vectors, the weights are all 3D orthogonal matrices. The difference is that matrix operation is used in neural networks for 3DV-BP while vector product operation is used in neural networks for the new algorithm. We expect that VP-BP can be effectively used in the field dealing with three-dimensional vectors, especially vector product operation. This new algorithm was applied to the XOR problem. Results suggest that the new method is superior to standard BP [5].

2 THE "VP-BP" ALGORITHM

2.1 A Vector Product Neuron

There appear to be several approaches for extending the standard BP to higher dimensions. One approach is to extend the number field, i.e. from real numbers x (1 dimension), to complex numbers $z = x + iy$ (2 dimensions; [1], [2], [3]), to quaternions $q = a + ib + jc + kd$ (4 dimensions), to sedenions (16 dimensions), $\cdots$. Another approach is to extend the dimensionality of the weights and threshold values from 1 dimension to n dimensions using n-dimensinal real valued vectors. Moreover, the latter approach has two varieties : (a) weights are n-dimensional matrices [4], (b) weights are n-dimensional vectors. In this paper we use the approach (b) ($n = 3$), in which vector product is adopted for the multiplication of vectors.

A model neuron used in the VP-BP algorithm is as follows. The input signals, weights, thresholds and output signals are all 3D real valued vectors. The activity $\boldsymbol{A}_j$ (analogous to the real activity in the standard BP) of neuron j is defined to be :

$$\boldsymbol{A}_j = \sum_k (\boldsymbol{W}_{jk} \times \boldsymbol{S}_k) + \boldsymbol{T}_j, \tag{1}$$

where $\boldsymbol{W}_{jk}$ is the 3D real valued vector weight connecting neuron j and k, $\boldsymbol{S}_k$ is the 3D real valued vector input signal coming from the output of neuron k, $\boldsymbol{T}_j$ is the 3D real valued vector threshold value of neuron j, and $\boldsymbol{x} \times \boldsymbol{y}$ denotes vector product of $\boldsymbol{x} = {}^t[x_1 \;\; x_2 \;\; x_3]$ and $\boldsymbol{y} = {}^t[y_1 \;\; y_2 \;\; y_3]$, i.e. $\boldsymbol{x} \times \boldsymbol{y} = {}^t[x_2y_3 - x_3y_2 \;\; x_3y_1 - x_1y_3 \;\; x_1y_2 - x_2y_1]$. To obtain the (3D real valued vector) output signal, convert the activity value $\boldsymbol{A}_j$ into its three components as follows.

$$\boldsymbol{A}_j = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}. \tag{2}$$

The output signal $F(\boldsymbol{A}_j)$ is defined to be

$$F(\boldsymbol{A}_j) \;=\; \begin{bmatrix} f(a_1) \\ f(a_2) \\ f(a_3) \end{bmatrix}, \qquad \text{where} \quad f(a_i) = \frac{1}{1 + \exp(-a_i)}. \tag{3}$$

2.2 A Vector Product Neural Network

In this subsection, we introduce the network used in the VP-BP algorithm. It has 3 layers, for the sake of simplicity. We use $\boldsymbol{W}_{ji} = {}^t[w_{ji}^x \;\; w_{ji}^y \;\; w_{ji}^z] \in \boldsymbol{R}^3$ for the weight between the input neuron i and the hidden neuron j (where $\boldsymbol{R}$ denotes the set of real numbers), $\boldsymbol{V}_{kj} = {}^t[v_{kj}^x \;\; v_{kj}^y \;\; v_{kj}^z] \in \boldsymbol{R}^3$ for the weight between the hidden neuron j and the output neuron k, $\boldsymbol{\Theta}_j = {}^t[\theta_j^x \;\; \theta_j^y \;\; \theta_j^z] \in \boldsymbol{R}^3$ for the threshold of the hidden neuron j, $\boldsymbol{\Gamma}_k = {}^t[\gamma_k^x \;\; \gamma_k^y \;\; \gamma_k^z] \in \boldsymbol{R}^3$ for the threshold of the output neuron k. Let $\boldsymbol{I}_i = {}^t[I_i^x \;\; I_i^y \;\; I_i^z] \in \boldsymbol{R}^3$ denote the input signal to the input neuron i, and let $\boldsymbol{H}_j = {}^t[H_j^x \;\; H_j^y \;\; H_j^z] \in \boldsymbol{R}^3$ and $\boldsymbol{O}_k = {}^t[O_k^x \;\; O_k^y \;\; O_k^z] \in \boldsymbol{R}^3$ denote the output signals of the hidden neuron j, and the output neuron k, respectively. Let $\boldsymbol{\Delta}_k = {}^t[\delta_k^x \;\; \delta_k^y \;\; \delta_k^z] = \boldsymbol{T}_k - \boldsymbol{O}_k \in \boldsymbol{R}^3$ denote the error between $\boldsymbol{O}_k$ and the target output signal $\boldsymbol{T}_k = {}^t[T_k^x \;\; T_k^y \;\; T_k^z] \in \boldsymbol{R}^3$ of the pattern to be learned for the output neuron k. We define the square error for the pattern p as $E_p = (1/2)\sum_{k=1}^{N} ||\boldsymbol{T}_k - \boldsymbol{O}_k||^2$, where N is the number of output neurons, $||\boldsymbol{x}|| \overset{\text{def}}{=} \sqrt{x_1^2 + x_2^2 + x_3^2}$, $\boldsymbol{x} = {}^t[x_1 \;\; x_2 \;\; x_3]$.

2.3 The Learning Algorithm

Next, we define a learning rule for the VP-BP model described above. For a sufficiently small learning constant $\varepsilon > 0$, and using a steepest descent method, we can show that the weights and the thresholds should be modified according to the following equations.

$$\Delta \boldsymbol{V}_{kj} \overset{\text{def}}{=} \begin{bmatrix} \Delta v_{kj}^x \\ \Delta v_{kj}^y \\ \Delta v_{kj}^z \end{bmatrix} = -\,\varepsilon \begin{bmatrix} \frac{\partial E_p}{\partial v_{kj}^x} \\ \frac{\partial E_p}{\partial v_{kj}^y} \\ \frac{\partial E_p}{\partial v_{kj}^z} \end{bmatrix}, \quad \Delta \boldsymbol{\Gamma}_k \overset{\text{def}}{=} \begin{bmatrix} \Delta \gamma_k^x \\ \Delta \gamma_k^y \\ \Delta \gamma_k^z \end{bmatrix} = -\,\varepsilon \begin{bmatrix} \frac{\partial E_p}{\partial \gamma_k^x} \\ \frac{\partial E_p}{\partial \gamma_k^y} \\ \frac{\partial E_p}{\partial \gamma_k^z} \end{bmatrix},$$

$$\Delta \boldsymbol{W}_{ji} \overset{\text{def}}{=} \begin{bmatrix} \Delta w_{ji}^x \\ \Delta w_{ji}^y \\ \Delta w_{ji}^z \end{bmatrix} = -\,\varepsilon \begin{bmatrix} \frac{\partial E_p}{\partial w_{ji}^x} \\ \frac{\partial E_p}{\partial w_{ji}^y} \\ \frac{\partial E_p}{\partial w_{ji}^z} \end{bmatrix}, \quad \Delta \boldsymbol{\Theta}_j \overset{\text{def}}{=} \begin{bmatrix} \Delta \theta_j^x \\ \Delta \theta_j^y \\ \Delta \theta_j^z \end{bmatrix} = -\,\varepsilon \begin{bmatrix} \frac{\partial E_p}{\partial \theta_j^x} \\ \frac{\partial E_p}{\partial \theta_j^y} \\ \frac{\partial E_p}{\partial \theta_j^z} \end{bmatrix}, \qquad (4)$$

where Δx denotes the amount of the correction of a parameter x. The above equations (4) can be expressed as :

$$\begin{aligned} \Delta \boldsymbol{V}_{kj} &= \boldsymbol{H}_j \times \Delta \boldsymbol{\Gamma}_k, & (5) \\ \Delta \boldsymbol{\Gamma}_k &= \varepsilon \boldsymbol{C}_k \boldsymbol{\Delta}_k, & (6) \\ \Delta \boldsymbol{W}_{ji} &= \boldsymbol{I}_i \times \Delta \boldsymbol{\Theta}_j, & (7) \\ \Delta \boldsymbol{\Theta}_j &= \boldsymbol{D}_j \sum_k (\Delta \boldsymbol{\Gamma}_k \times \boldsymbol{V}_{kj}), & (8) \end{aligned}$$

where

$$\boldsymbol{C}_k = \begin{bmatrix} (1-O_k^x)O_k^x & 0 & 0 \\ 0 & (1-O_k^y)O_k^y & 0 \\ 0 & 0 & (1-O_k^z)O_k^z \end{bmatrix}, \qquad (9)$$

$$\boldsymbol{D}_j = \begin{bmatrix} (1-H_j^x)H_j^x & 0 & 0 \\ 0 & (1-H_j^y)H_j^y & 0 \\ 0 & 0 & (1-H_j^z)H_j^z \end{bmatrix}. \qquad (10)$$

3 SIMULATION

The XOR problem was used to compare the performance of the new VP-BP algorithm with the standard back-propagation algorithm. We used a 1-3-1 three-layered network for the VP-BP, and a 3-7-3 three-layered network for the standard BP. Table 1 shows that their time complexities per learning cycle are almost equal. The learning constant used in the experiment was 0.5. The initial X-, Y- and Z-components of the weights and the thresholds were chosen to be random real numbers between − 0.3 and + 0.3. The input data are presented in sequence, together with the desired output, to the net as shown in Table 2.

The results of the simulation are plotted in Fig.1. The new algorithm converged in 1,500 iterations, whereas the original algorithm required 3,000. Furthermore, the space complexity (i.e. the number of parameters) is almost half that of the standard BP, as seen in Table 1.

4 CONCLUSIONS

We have proposed a three-dimensional vector version of the back-propagation learning algorithm, for neural networks based on vector product, where the input signals, weights, thresholds, and output signals are all 3D real valued vectors. The XOR problem was used to test the presented method and it showed excellent performance. We expect that this new algorithm will demonstrate its real ability in the areas dealing with three-dimensional vectors, especially vector product operation. The extension of the VP-BP algorithm to fully connected neural networks will be presented in a future paper. It seems that higher dimensional (more than 4 dimensions) version of the back-propagation learning algorithm can be derived using higher dimensional vector product.

ACKNOWLEDGEMENTS
The author would like to express his thanks to Dr.T.Yuba, Director of the Computer Science Division, and Dr.T.Furuya, Chief of the Computational Models Section, for having an opportunity to do this study and their continual encouragement.

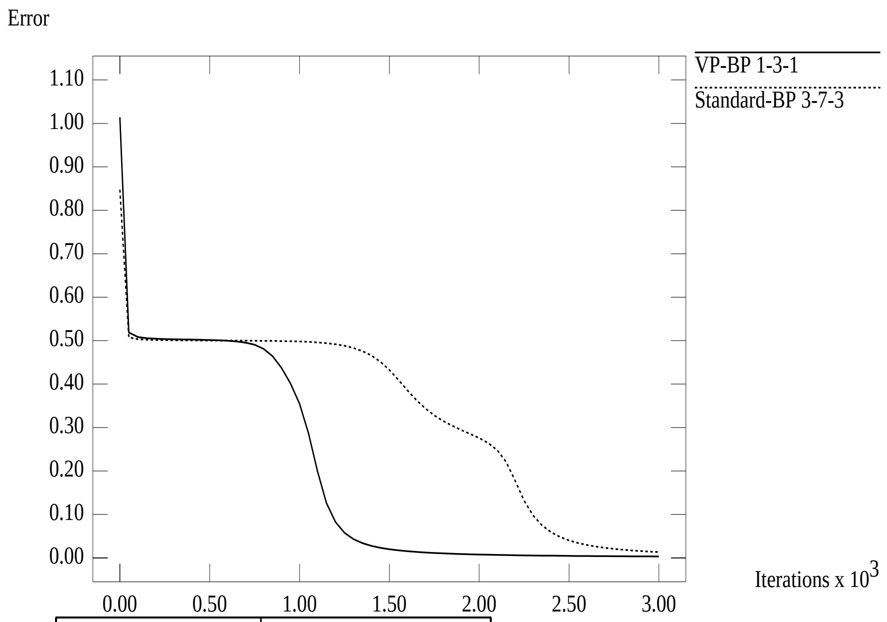
References

[1] M. S. Kim and C. C. Guest, "Modification of backpropagation networks for complex-valued signal processing in frequency domain," in *Proc. IJCNN*, Vol.3, June 1990, pp.27-31.
[2] T. Nitta and T. Furuya, "A complex back-propagation learning and the capability for the transformation of geometrical figures," Tech. Rep. of IEICE, NC90-60, pp. 45-52, Feb. 1991 (in Japanese).
[3] T. Nitta and T. Furuya, "A complex back-propagation learning," *Transactions of Information Processing Society of Japan*, Vol.32, No.10, pp. 1319-1329, 1991 (in Japanese).
[4] T. Nitta and H. de Garis, "A 3D vector version of the back-propagation algorithm," in *Proc. IJCNN*, Vol.2, Nov. 1992, pp. 511-516.
[5] D. E. Rumelhart et al., *Parallel Distributed Processing*, Vol.1, MIT Press, 1986.

Network	Time complexity			Space complexity		
	$\times$ and $\div$	$+$ and $-$	Sum	Weights	Thresholds	Sum
VP-BP 1-3-1	117	78	195	18	12	30
Standard BP 3-7-3	128	69	197	42	10	52

Table 1 : The Computational Complexity of the VP-BP and the Standard BP. Time complexity means the sum of the four operations performed per learning cycle. Space complexity means the sum of the parameters (weights and thresholds).

Figure 1: Learning Curves for the XOR problem.



Input			Output		
x	y	z	x	y	z
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	1	0	0
1	1	0	0	0	0

Table 2 : The Input Patterns and the Corresponding Desired Output Patterns for the XOR Problem. x is given to the X-component of the input/output neuron 1, y is the Y-component, and z is the Z-component in the 3DV-BP network. x is given to the input/output neuron 1, y is the neuron 2, and z is the neuron 3 in the standard BP network.