Proceedings of 1993 International Joint Conference on Neural Networks

INTERVAL ARITHMETIC BACKPROPAGATION

C.A. Hernández [1], J. Espí [1], K. Nakayama [3], M. Fernández[1].

[1] Department of Computers and Electronics.
Valencia University.
Doctor Moliner, 50.
46100 Burjassot (Valencia).
SPAIN
E-mail: HERNANDC@EVALUN11.BITNET

[3] Department of Electrical and Computers
Engineering. Kanazawa University.
Kodatsuno 2-20-40. Kanazawa-shi.
Ishikawa-ken. 920 JAPAN
E-mail:
Nakayama@haspnn1.ec.kanazawa-u.ac.jp

Abstract

We present a new extension of the Backpropagation learning algorithm by using interval arithmetic. The proposed algorithm represents a generalization of Backpropagation and contains Backpropagation like a particular case. This new algorithm permits the use of training samples and targets which can be indistinctly points and intervals.

Among the possible applications of this algorithm, we report its usefulness to integrate expert's knowledge and experimental samples and also its ability to handle "don't care attributes" in a simple and natural way in comparison with Backpropagation. It also adds flexibility to the codification of inputs and outputs.

1.- Introduction.

Backpropagation with Multilayers Neural Networks was one of the first successful neural networks training paradigms [1] and it is also one of the most used nowadays.

There have been many modifications of this training algorithm. One of the useful ones, under the point of view of the authors, is the extension to interval arithmetic proposed in [2]. Interval arithmetic [3] allows Backpropagation to combine real vectors and interval vectors as training samples and targets for a neural network. In this way an interval at the input of the neural network is converted to another interval at the output.

However, a quite severe limitation of the algorithm proposed in [2] is that it has only one output unit and can only be applied to classification problems with two classification classes. Furthermore, it does not constitute a generalization of Backpropagation because of the definition of its cost function.

In this paper we propose a new and direct extension of Backpropagation to interval arithmetic which is called Interval Arithmetic Backpropagation (IABP). This new algorithm can be used with any number of output units, every equation reduces to the equations of Backpropagation for the case of a real vector input and under this point of view IABP can be considered a generalization of BP.

We show that this algorithm can integrate expert's knowledge and training samples in the training set in the same way of [2] and we also show that it can handle "don't care attributes" in a very simple and advantageous way in comparison with Backpropagation [4]. We finally discuss that the use of intervals in the input can add flexibility to the codification.

2.- Interval Arithmetic Backpropagation.

The basic equations of interval arithmetic [2,3] which are useful in the following development are:

Sum of intervals:

$$A + B = [a^L, a^U] + [b^L, b^U] = [a^L+b^L, a^U+b^U]$$

Product by a real number:

$$m \cdot A = m \cdot [a^L, a^U] = \begin{cases} [m \cdot a^L, m \cdot a^U] \ , & if \ \ m \geq 0 \\ [m \cdot a^U, m \cdot a^L] \ , & if \ \ m < 0 \end{cases}$$

Exponential function:

$$\exp A = \exp\, [a^L, a^U] = [\exp a^L, \exp a^U]$$

The superscript L denotes the lower limit of the interval and the superscript U the upper one. Now, we will define a generalization to interval arithmetic of the neuron transfer function:

$$f(Net) = f([net^L, net^U]) = [f(net^L), f(net^U)]$$

where:

$$f(x) = \frac{1}{1 + \exp(-x)}$$

This definition is consistent because f(x) increases monotonically. Next, we will define the relationships in the neural network.

The input patterns will be, in general, interval vectors:
$$I_{P,i} = [I^L_{P,i}, I^U_{P,i}]$$

The output of the hidden units:

$$H_{P,j} = f(Net_{P,j}) \qquad where: \qquad Net_{P,j} = \sum_{i=1}^{Ninputs} w_{j,i} \cdot I_{P,i} + \theta_j$$

and:

$$net^L_{P,j} = \sum_{i=1,\, w_{j,i} \geq 0}^{Ninputs} w_{j,i} I^L_{P,i} + \sum_{i=1,\, w_{j,i} < 0}^{Ninputs} w_{j,i} I^U_{P,i} + \theta_j$$

$$net^U_{P,j} = \sum_{i=1,\, w_{j,i} \geq 0}^{Ninputs} w_{j,i} I^U_{P,i} + \sum_{i=1,\, w_{j,i} < 0}^{Ninputs} w_{j,i} I^L_{P,i} + \theta_j$$

The outputs of the neural network are defined in the same way for the weights $w_{k,i}$.

The targets will also be, in general, interval vectors:
$$T_{P,k} = [t^L_{P,k}, t^U_{P,k}]$$

And the mean square error function can be defined as a generalization of the BP error function:

$$E_P = \frac{1}{4} \cdot \sum_{k=1}^{Noutputs} ((t^L_{P,k} - O^L_{P,k})^2 + (t^U_{P,k} - O^U_{P,k})^2)$$

The learning in IABP is the process of minimizing the above error function, the weights are changed according to the following function:

$$\Delta w_{j,i}(t+1) = \eta \cdot \left(-\frac{\partial E_P}{\partial w_{j,i}}\right) + \beta \cdot \Delta w_{j,i}(t)$$

where η is the step size and β the momentum. The partial derivatives can be calculated as in Backpropagation. The result for the weights between the hidden units and the output is the following:

$$\frac{\partial E_P}{\partial w_{k,i}} = \begin{cases} \delta^L_{P,k} H^L_{P,i} + \delta^U_{P,k} H^U_{P,i} \ , & if \ \ w_{k,i} \geq 0 \\ \delta^L_{P,k} H^U_{P,i} + \delta^U_{P,k} H^L_{P,i} \ , & if \ \ w_{k,i} < 0 \end{cases}$$

where:

$$\delta^{L}_{P,k} = -\frac{1}{2}\ (t^{L}_{P,k}-O^{L}_{P,k})\ O^{L}_{P,k}\ (1-O^{L}_{P,k})$$

$$\delta^{U}_{P,k} = -\frac{1}{2}\ (t^{U}_{P,k}-O^{U}_{P,k})\ O^{U}_{P,k}\ (1-O^{U}_{P,k})$$

And for the weights between the hidden units and the input:

$$\frac{\partial E_P}{\partial w_{j,i}} = \begin{cases} \delta h^{L}_{P,j} I^{L}_{P,i} + \delta h^{U}_{P,j} I^{U}_{P,i}\ , if\ w_{j,i} \geq 0 \\ \delta h^{L}_{P,j} I^{U}_{P,i} + \delta h^{U}_{P,j} I^{L}_{P,i}\ , if\ w_{j,i} < 0 \end{cases}$$

where:

$$\delta h^{L}_{P,j} = \Big(\sum_{k,\,w_{k,j}\geq 0} \delta^{L}_{P,k} \cdot w_{k,j} + \sum_{k,\,w_{k,j}<0} \delta^{U}_{P,k} \cdot w_{k,j} \Big) \cdot H^{L}_{P,j} \cdot (1-H^{L}_{P,j})$$

$$\delta h^{U}_{P,j} = \Big(\sum_{k,\,w_{k,j}\geq 0} \delta^{U}_{P,k} \cdot w_{k,j} + \sum_{k,\,w_{k,j}<0} \delta^{L}_{P,k} \cdot w_{k,j} \Big) \cdot H^{U}_{P,j} \cdot (1-H^{U}_{P,j})$$

The percentage correct error function can also be defined for classification problems. A possible definition with threshold 0.5 is: suppose the target for a pattern P is $T_j=[1,1]$ and $T_i=[0,0]$ for $i \neq j$ and $i=1...$Noutput. Then, this pattern should count positively in the percentage if $O_{P,j}{}^{L} \geq 0.5$ and $O_{P,i}{}^{U} < 0.5$.

3.- Experimental results.

We present three bidimensional examples, Figs.1, 2, 3. The example in Fig.1 corresponds to a mixture of real and interval vectors in the training set, the real vectors are codified as closed interval vectors with only one point inside each interval, Table 1. In Fig.1 it is represented the training samples and a line which represents the threshold 0.5 of the output units, i.e., the achieved classification. This result corresponds to a neural network with 2 output units, 5 hidden units and 2 inputs. The example in Fig.2 corresponds to the results of a neural network with 2 inputs, 7 hidden units and 3 outputs, the training set is composed of interval vectors which are represented in the figure by rectangles. Finally, example 3 corresponds to a neural network with 2 inputs, 8 hidden units and 5 outputs. The input set is the same of example 2, but the classification classes are different.

It is achieved a perfect classification of the training set in all the examples.

3.1.- Integration of expert's knowledge and sample data.

The kind of expert's knowledge considered is a set of "if ...then" rules like the following one:

if $X_{P,1} \subset [A_1, B_1]$ ... and $X_{P,n} \subset [A_n, B_n]$ then $X_P \in G_K$

where X_P is a pattern vector and G_K its classification class.

<u>This type of rules can be easily codified by using interval arithmetic</u>, the set of intervals $[A_i, B_i]$ will be used like the input of the neural network $\{[A_1, B_1], [A_2, B_2] \ldots [A_n, B_n]\}$ and the corresponding target will be the codification of the class G_k. After that, this kind of intervals can be included in the training set together with sample data, a clear example is example 1.

3.2.- Handle of "don't care attributes".

The definition of a "don't care attribute" and a research in the codification for normal BP in the case of discrete inputs

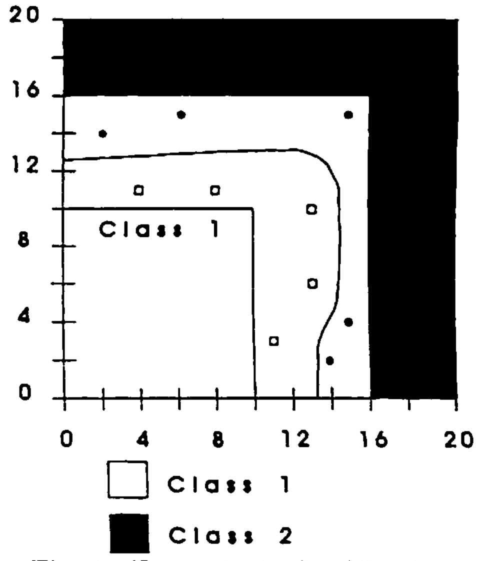

Fig. 1. Example 1, classification.

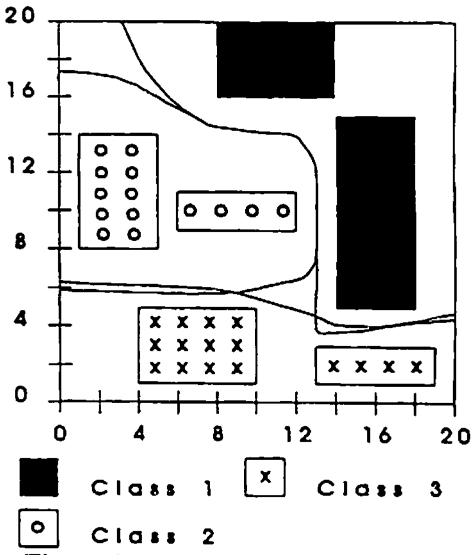

Fig.2. Example 2, classification.

can be found in [4]. Table 2 corresponds to the second example in that paper. Suppose we have the sample vector: [1, D, D, D, D, -1] (last row in Table 2) where D means "don't care" or in other words "whatever the value of" this input. According to the results in [4], for normal BP we should include in the training set all the vectors which result from codifying D with all its possible values or with two values: the maximum possible value d_{max} and the minimum one d_{min}. This yields to an exponential increase in the training set if we have more than one D in a vector. In the case of the last sample vector, we should use 81 training samples if we codify it with all its possible values and 16 if we codify with the maximum and minimum value. With interval arithmetic we can codify D like an interval $[d_{min}, d_{max}]$, the result is only one training sample and this approach is valid for the discrete and the continuous case. We have reproduced the best result obtained in [4] for the three examples used there (IS1, IS2 and IS3) by using an interval arithmetic codification of the "don't care attributes" and IABP. The training set in our case was smaller and also the number of hidden units of the neural network.

Table 1. Training set of example 1.

Class.	Training Samples.	Target
C.1 X	[4,4] [8,8] [11,11] [13,13] [13,13] [0,10]	[1,0]
C.1 Y	[11,11] [11,11] [3,3] [6,6] [10,10] [0,10]	[1,0]
C.2.Y	[14,14] [15,15] [2,2] [4,4] [15,15] [0,20] [16,20]	[0,1]
C.2 X	[2,2] [6,6] [14,14] [15,15] [15,15] [16,20] [0,20]	[0,1]

Table 2. Training instances IS2.

Worth	Employee acceptance	Solution available	Easier solution	Teachability	Risk	Suitability
1	1	2	2	-1	2	0
-1	-1	-1	1	2	-1	0
-1	2	2	-1	2	-1	1
-1	2	-1	2	-1	1	0
-1	1	2	2	2	-1	1
-1	1	-1	-1	-1	-1	1
1	2	2	-1	-1	2	0
1	-1	-1	1	-1	2	1
2	D	D	D	D	D	1
1	D	D	D	D	-1	1

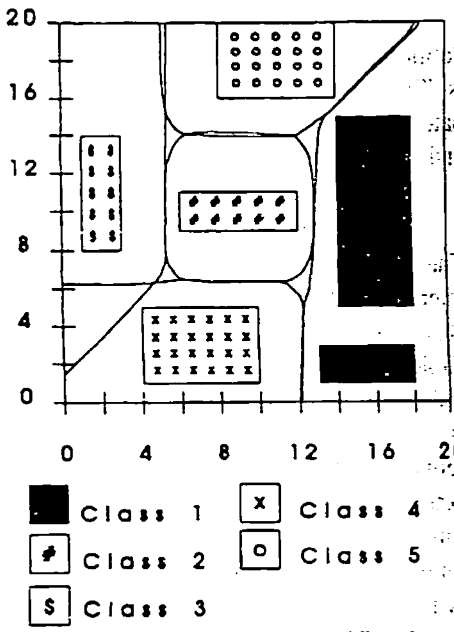

Fig. 3. Example 3, classification.

4.- Conclusion and Discussion.

We have generalized the BP training algorithm to interval arithmetic and we have shown two possible applications of this new algorithm: integration of expert's knowledge and sample data and the handle of "don't care attributes". In general, this algorithm will add flexibility to the codification of inputs and targets. For example, in the case we have a strong subjectivity and imprecision (e.g., the codification of symptoms in a medical diagnosis classification problem) the use of intervals in the codification may reduce this subjectivity and imprecision, an imprecise or subjective input could be codify with a wider interval. Perhaps, it would be a more appropriate way to codify these cases.

5.- References.

[1] D.E. Rumelhart, J.L. McClelland and the PDP Research Group. *Parallel Distributed Processing* Vol. MIT Press, Cambridge, 1988.

[2] H. Ishibuchi and H. Tanaka. "An extension of the BP-Algorithm to Interval Input Vectors -Learning from Numerical Data and Expert's Knowledge-". IJCNN-91. Singapoure. pp. 1588-1593.

[3] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press. New York.

[4] H.M. Lee and C. Hsu. "The Handling of Don't Care Attributes". IJCNN-91. Singapoure. pp.1085-