3 HASEGAWA, T., BANBA, S., OGAWA, H., and NAKAMOTO, H.: 'Characteristics of valley microstrip lines for use in multilayer MMICs', *IEEE Microw. Guid. Wave Lett.*, 1991, **1**, (10), pp. 275–277
4 GILLICK, M., and ROBERTSON, I.D.: 'An X-band monolithic power amplifiers using low characteristic impedance thin-film microstrip transformers', *IEEE Microw. Guid. Wave Lett.*, 1992, pp. 328–330
5 GILLICK, M., and ROBERTSON, I.D.: 'Ultra low impenance CPW transmission lines for multilayer MMICs'. IEEE MTT-S Int. Microwave Symp. Dig., June 1993, pp. 127–130
6 LUCYSZYN, S., WANG, Q.H., and ROBERTSON, I.D.: '0.1THz rectangular waveguide on GaAs semi-insulating substrate', *Electron. Lett.*, 1995, **31**, pp. 721–722

Incorporating *a priori* knowledge into neural networks

D. Anguita, S. Ridella, S. Rovetta and R. Zunino

Indexing terms: Neural networks, Backpropagation, Multilayer perceptrons, Pattern classification

A correct interval arithmetic back-propagation (IABP) algorithm is derived that allows one to incorporate some *a priori* knowledge into a neural classifier. The proposed method also allows a straightforward circuit implementation.

Introduction: The process of building a reliable classifier from a set of samples by using a multilayer perceptron (MLP) is not trivial and in many cases could be very helpful to embody in the network some simple *a priori* knowledge (if any) of the problem. We can encode *a priori* knowledge expressed as rules of the kind:

$$\text{IF } x_i^p \in [\underline{X}_1^p, \overline{X}_1^p] \text{ AND } \ldots$$
$$\text{AND } x_N^p \in [\underline{X}_N^p, \overline{X}_N^p] \text{ THEN } \mathbf{x}^p \in C_k$$

where x_i^p is the ith component of the pattern $\mathbf{x}^p$ and C_k is one of the possible classes. The intervals $[\underline{X}_i^p, \overline{X}_i^p]$ define a region of the input space (a hyperrectangle) whose points are known to belong to a particular class. A multilayer perceptron that makes use of interval arithmetic [1] (IAMLP) has been proposed for the purpose of computing with interval values [2]. The IAMLP can be fed directly with the intervals defined by the set of rules and, at the same time, act like a conventional MLP: this can be obtained by reducing the hyperrectangle to a point, letting the two extremes of the interval coincide.

Interval arithmetic back-propagation: Let us consider two intervals X and Y, and a real number w. We can define arithmetic operations $(+, -, *)$ and the sigmoid function sgm() as follows:

$$X + Y = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}]$$
$$X - Y = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}]$$
$$\text{sgm}(X) = [\text{sgm}(\underline{X}), \text{sgm}(\overline{X})]$$
$$wX = [w\underline{X} \cdot H(w) + w\overline{X} \cdot H(-w),$$
$$w\overline{X} \cdot H(w) + w\underline{X} \cdot H(-w)]$$

where H is the Heaviside function.

By using the rules of interval arithmetic, one could derive IABP, at least in principle, in the same way as conventional BP. Unfortunately, the resulting algorithm does not ensure that it will follow a descent direction. In fact, the peculiar form of the product causes a first-order discontinuity of the gradient for $w = 0$.

Here we propose a solution to this problem. We approximate the interval product by a differentiable version, by replacing the Heaviside function with a sigmoid: $H(w) \simeq \text{sgm}(\gamma w)$, where γ controls the steepness of the sigmoid. Note that if $\gamma \to \infty$ or the interval degenerates to a single point ($\underline{X} = \overline{X}$), the two versions of the product coincide.

Furthermore, we define an error function that takes into account the fact that, in general, the outputs are interval values:

$$E = \frac{1}{2} \sum_p \sum_i \left[(t_i^p - \underline{O}_i^p)^2 + (t_i^p - \overline{O}_i^p)^2 \right]$$

Thanks to the proposed approximation, the IABP algorithm can now be easily derived in the same way as conventional BP by differentiating E with respect to the weights of the network (the formulas are omitted due to lack of space).

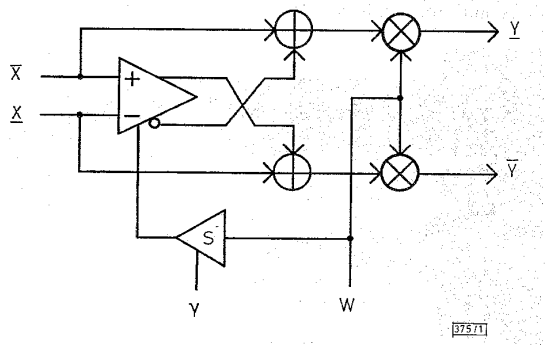

Fig. 1 *Synapse of IAMLP*

Fig. 1 shows the circuit that implements one of the synapses of the IAMLP. It computes the product between a weight w and the input interval X, making use of an amplifier (S) with sigmoidal transfer function. The activation of the neuron (not shown) can be computed separately, in a conventional way, on the two extremes of the output interval Y.

During learning, in order to reproduce the correct behaviour of the IAMLP, the parameter γ should be increased, by using, for instance, an annealing process [3]. Note, however, that the weights of the network grow during the learning phase and tend to push the network behaviour towards the correct one. In fact, during our experiments we simply raised γ in the final learning phases by doubling its magnitude at each step.

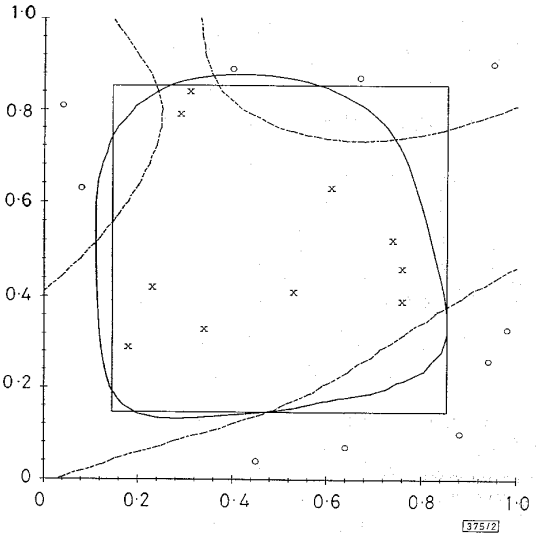

Fig. 2 *Comparison between MLP and IAMLP*
– – – MLP
——— IAMLP

IAMLP learns from a priori knowledge: Here we present an example of the ability of the IAMLP to use some *a priori* knowledge to learn a simple problem. Our training set is composed of 20 random points sampled in a unit square: the points lying inside a square of area 1/2 and centred in (0.5,0.5) belong to class C_1, and the others to class C_0 (Fig. 2). The discriminating functions generated by a conventional MLP and by an IAMLP are also shown. The MLP can correctly classify all the points of the training set, but fails to learn the underlying problem. On the contrary, the IAMLP grabs the main characteristic of the problem. The additional information provided to the IAMLP states that all the points at the border of the problem space belong to class C_0 (i.e. IF $X_i = 0$ OR $X_i = 1$ THEN $(X_1, X_2) \in C_0$). Note that, to provide the same information to the MLP, it would be necessary to sample the space defined by the IF clause with a large number of points: this would increase the learning complexity exponentially for high-dimensional problems.

Conclusions: The proposed IAMLP is able to learn some *a priori* knowledge (i.e. IF THEN rules) in addition to the patterns that can be learned by a conventional MLP. This is accomplished by converting the rules to interval values that can be supplied directly to the network. Furthermore, the IAMLP allows a straightforward circuit implementation and the derivation of IABP, a BP-like learning algorithm.

3 August 1995
Electronics Letters Online No: 19951309

D. Anguita, S. Ridella, S. Rovetta and R. Zunino (*University of Genova, Department of Biophysical and Electronic Engineering, Via Opera Pia 11A, I-16145 Genova, Italy*)

References

1 ALEFELD, G., and HERZBERGER, J.: 'Introduction to interval computation' (Academic Press, New York, 1983)
2 ISHIBASHI, H., and TANAKA, H.: 'An extension of the BP-algorithm to interval input vectors – learning from numerical data and expert's knowledge'. Proc. Int. Joint Conf. on NN, Singapore, 18–21 November 1991, pp. 1588–1593
3 YU, X., LOH, N.K., and MILLER, W.C.: 'Training hard-limiting neurons using back-propagation algorithm by updating steepness factor'. Proc. IEEE Conf. on Neural Networks, Orlando, FL, USA, 1994, pp. 526–530

Increasing innate robustness in artificial neural networks using redundancy

M.P. Thompson and C. Kambhampati

Indexing terms: Neural networks, Redundancy, Fault tolerant computing

A theoretical explanation of robustness and its relationship with redundancy is proposed and used to derive a novel and powerful technique which allows the innate robustness of most types of artificial neural network (ANN) to be enhanced to a user-defined degree.

Introduction: ANNs are models of biological neural networks which continue to function despite individual cell deaths due to their innate (or on-line) robustness. The ANN equivalent of cell death, namely neuron removal, how it can be understood and countered, is considered in this Letter. Robustness can be engineered into an ANN either during or after training. Techniques for increasing robustness during training include training a network known to have too large a topology (which has the disadvantage of inferior generalising ability) and/or altering the network's training regime to explicitly consider robustness [1, 2]. Techniques for increasing robustness after training include: (i) majority voting with x identical networks, (ii) majority voting with x differently trained networks [3], (iii) increasing reliability of each neuron through statistical means [4], and (iv) intrinsic fault tolerance enhanced by redundancy [5]. The proposed theory and technique, like [5], rely on the intrinsic summation of weighted inputs in neurons, and are applicable to any ANN model which uses such neurons.

Definitions: $\lambda(\gamma,i)$ is the ith element of Λ_γ: the of set of neurons outputting to neuron γ. The maximum absolute contribution (MAC) of neuron a to neuron b is denoted $\delta_{ab} = |w_{ab}.m_a|$, where w_{ab} is the weight from neuron a to neuron b, and m_a is the maximum absolute output of neuron a. Heaviside(x) = 1 when $x \geq 0$ and –1 when $x < 0$. x^- is infinitesimally less than x. x^+ is infinitesimally more than x. The bth copy of neuron a is denoted by ${}^b a$.

Theory: It is proposed that each neuron γ has an intrinsic capacity $\beta_\gamma^{cap}(\geq 0)$ for absorbing errors in its sum of weighted inputs, which is the lesser of infinitely less than: (i) $\beta_\gamma^\downarrow$ the minimum decrease in γ's bias β_γ, and (ii) $\beta_\gamma^\uparrow$ the minimum increase in γ's bias β_γ, which alters γ's output behaviour. β_γ^{cap} can easily be derived from the input/output behaviour of γ found when calculating the deviation between actual and desired network outputs before any damage. Each neuron $\lambda(\gamma, i)$ can alter γ's sum of weighted inputs by up to its MAC. This MAC is the maximum possible error in γ's inputs that can occur directly as a result of $\lambda(\gamma, i)$ being destroyed. Thus, in a noiseless system, so long as the sum of the MACs of neurons removed from Λ_γ is $\leq \beta_\gamma^{cap}$, γ's output cannot be altered by those removals:

$$\sum_{\forall i} \overbrace{\text{error in output of } \lambda(\gamma,i)}^{\text{in noiseless case}=\delta_{\lambda(\gamma,i)\gamma}} \leq \beta_\gamma^{cap} \Rightarrow \gamma\text{'s output unaffected} \quad (1)$$

If eqn. 1 is false, then the removal(s) may corrupt γ's output. If a corrupted neuron corrupts one or more others which in turn corrupt others, then the number of erroneous neuron outputs can increase quickly, in a process analogous to a chain reaction. In this way a neuron which can tolerate erroneous inputs from a small number of neurons outputting directly to it can be corrupted by a single corrupt neuron outputting indirectly to it. If an output neuron's behaviour is altered, then this will alter the classification performance of the network as a whole.

The question arises of how many neurons can be removed from a network without compromising classifying performance. The most obvious way of calculating whether or not particular damage to a particular network affects classification performance on a particular data set is simply to calculate the deterioration in closeness of fit on that data due to network damage.

In the limited noiseless case where neuron γ takes inputs from neurons with equal MACs ($\delta_{\lambda(\gamma,i)\gamma} = \delta_\gamma \; \forall \; i$), eqn. 1 can be used to derive the maximum number of neurons outputting to γ which may be removed without risk of altering γ's output characteristics (eqn. 2):

$$\text{Safely removable } \Lambda_\gamma \text{ upper bound} = \text{integer part of } \left(\frac{\beta_\gamma^{cap}}{\delta_\gamma}\right) \quad (2)$$

Eqns. 1 and 2 show that as the ratio of β_γ^{cap} increases relative to the possible MACs of neurons outputing to γ, the less reliant γ is on each of them. The following technique increases this ratio.

Technique: Innate robustness can be increased for a neuron η and therefore in a network on a neuron-by-neuron basis using the steps below. The proposed technique has the effect of sharing information previously stored only in one neuron η, between c_η identical neurons each of which performs the same function as η but with proportionately less weight. Because each copy has less weight, a neuron γ inputting from η's copies is less dependent on each one due to a correspondingly smaller MAC. c_η is η's degree of redundancy (DoR).

Step 1: Maximise β_γ^{cap} if possible by altering γ's bias β_γ so that $\beta_\gamma^\downarrow$ and $\beta_\gamma^\uparrow$ are equal. This can be achieved by increasing β_γ by $0.5((\beta_\gamma^\uparrow - \beta_\gamma^\downarrow)^-)$. If γ has sigmoid thresholding which is made use of (which cannot be substituted with Heaviside thresholding without altering γ's input/output behaviour), then $\beta_\gamma^\downarrow$ and $\beta_\gamma^\uparrow$ will already be equal at 0+.

Step 2: Duplicate η c_η–1 times, making c_η identical copies of η. The copies ${}^i\eta$ ($1 \leq i \leq c_\eta$) should be connected to the same neurons, and with identical weights as the original η (both to and from η). The number of copies c_η depends on the degree of robustness required, and limits such as maximum number of neurons or maximum fan-in. The greater the number of copies, the greater the innate robustness.

Step 3: Every neuron γ inputting from what was η will now receive from ${}^i\eta$ ($1 \leq i \leq c_\eta$) inputs totalling exactly c_η times greater than from η (simple scaling). To maintain γ's sum of weighted inputs unchanged it is necessary to reduce the weights from the neurons ${}^i\eta$ to neuron γ by a factor of c_η. To make the duplications transparent with respect to the whole network, this step must be performed for all γ. The transparency requirement is given in eqn. 3.