A TRAINING METHOD FOR DISCRETE MULTILAYER NEURAL NETWORKS

G.D. Magoulas, M.N. Vrahatis*, T.N. Grapsa* and G.S. Androulakis*

Department of Electrical and Computer Engineering, University of Patras, GR-261.10, Patras, Greece. Email: magoulas@ee-gw.ee.upatras.gr

** Department of Mathematics, University of Patras, GR-261.10 Patras, Greece. Email: vrahatis—grapsa—gsa@math.upatras.gr*

In this contribution a new training method is proposed for neural networks that are based on neurons whose output can be in a particular state. This method minimises the well known least square criterion by using information concerning only the signs of the error function and inaccurate gradient values. The algorithm is based on a modified one–dimensional bisection method and it treats supervised training in networks of neurons with discrete output states as a problem of minimisation based on imprecise values.

Subject classification: AMS(MOS) 65K10, 49D10, 68T05, 68G05.

Keywords: Numerical analysis, imprecise function and gradient values, hard–limiting threshold units, feed forward neural networks, supervised training, back–propagation of error.

1 Introduction

Consider a Discrete Multilayer Neural Network (DMNN) consisting of L layers, in which the first layer denotes the input, the last one, L, is the output, and the intermediate layers are the hidden layers. It is assumed that the (l-1)–th layer has N_{l-1} units. These units operate according to the following equations :

$$net_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^{l-1,l} y_i^{l-1} + \theta_j^l, \qquad y_j^l = \sigma^l\left(net_j^l\right), \tag{1}$$

where net_j^l is the net input to the jth unit at the lth layer, $w_{ij}^{l-1,l}$ is the connection weight from the ith unit at the $(l-1)$–th layer to the jth unit at the lth layer, y_i^l denotes the output of the ith unit belonging to the lth layer, θ_j^l denotes the threshold of the jth unit at the lth layer, and σ is the activation function. In this paper we consider units where $\sigma(net_i^l)$ is a discrete activation function. We especially focus on units with two output states, usually called binary or hard–limiting units [1], i.e. $\sigma^l(net_j^l) =$ "*true*", if $net_j^l \geq 0$, and "*false*" otherwise.

Although units with discrete activation function have been superseded to a large extent by the computationally more powerful units with analog activation function, still DMNNs are important in that they can handle many of the inherently binary tasks that neural networks are used for. Their internal representations are clearly interpretable, they are computationally simpler to understand than networks with sigmoid units and provide a starting point for the study of the neural network properties. Furthermore, when using hard–limiting units we can understand better the relationship between the size of the network and the complexity of the training [2]. In [3] it has been demonstrated that DMNNs with only one hidden layer, can create any decision region that can be expressed as a finite union of polyhedral sets when there is one unit in the input layer. Moreover, artificially created examples were given where these networks create non convex and disjoint decision regions.

Finally, discrete activation functions facilitate neural network implementations in digital hardware and are much less costly to fabricate.

The most common feed forward neural network (FNN) training algorithm, the back–propagation (BP) [4] that makes use of the gradient descent, cannot be applied directly to networks of units with discrete output states, since discrete activation functions (such as hardlimiters) are non–differentiable. However, various modifications of the gradient descent have been presented [5, 6, 7]. In [8] an approximation to gradient descent, the so–called pseudo–gradient training method, was proposed. This method uses the gradient of a sigmoid as a heuristic hint instead of the true gradient. Experimental results validated the effectiveness of this approach. In this paper, we derive and apply a new training method for DMNNs that makes use of the gradient approximation introduced in [8]. Our method exploits the imprecise information regarding the error function and the approximated gradient, like the pseudo–gradient method does, but it has an improved convergence speed and has potential to train DMNNs in situations where, according to our experiments, the pseudo–gradient method fails to converge.

2 Problem Formulation and Proposed Solution

We consider units with two discrete output states and we shall use the convention f (or $-f$) for "false" and t (or $+t$) for "true", where f, t are real positive numbers and $f < t$, instead of the classical 0 and 1 (or -1, and $+1$). Real positive values prevent units from saturating, give to the logic "false" some power of influence over the next layer of the DMNN, and help the justification of the approximated gradient value which we shall employ.

First, let us define the error for a discrete unit as follows: $e_j(t) = d_j(t) - y_j^L(t)$, for $j = 1, 2, \ldots, N_L$, where $d_j(t)$ is the desired response at the jth neuron of the output layer at the input pattern t, $y_j^L(t)$ is the output at the kth neuron of the output layer L. For a fixed, finite set of input–output cases, the square error over the training set which contains T representative cases is:

$$E = \sum_{t=1}^{T} E(t) = \sum_{t=1}^{T} \sum_{j=1}^{N_L} e_j^2(t). \tag{2}$$

The idea of the pseudo–gradient was first introduced in training discrete recurrent neural networks [9, 10] and extended to DMNNs [8]. The method approximates the true gradient of the error function with respect to the weights, i.e. $\nabla E(w)$, by introducing an analog set of values for the outputs of the hidden layer units and the output layer units.

Thus, it is assumed that y_j^l in (1) can be written as $y_j^l = \tilde{\sigma}^l\Big(s(net_j^l)\Big)$, where $\tilde{\sigma}$(x)= "*true*" if x$\geq$ 0.5, and "*false*" otherwise, if $s(\cdot)$ is defined in $[0, 1]$. If $s(\cdot)$ is defined in $[-1, 1]$ then $\tilde{\sigma}$(x)= "*true*" if x$\geq$ 0, and "*false*" otherwise.

Using the chain rule, the pseudo–gradient is computed :

$$\frac{\widetilde{\partial E}}{\partial w_{ij}^{l-1,l}} = \widetilde{\delta_j^l} y_i^{l-1}, \tag{3}$$

where the back–propagating error signal $\tilde{\delta}$ for the output layer is $\widetilde{\delta_j^L} = \Big(d_j - s(net_j^L)\Big) \cdot s'(net_j^L)$ and for the hidden layers ($l \in [2, L-1]$) is $\widetilde{\delta_j^l} = s'(net_j^l)$

$\sum_n w_{jn}^{l,l+1} \widetilde{\delta_n^{l+1}}$. In these relations $s'(net_j^l)$ is the derivative of the analog activation function.

By using real positive values for "true" and "false" we ensure that the pseudo-gradient will not reduce to zero when the output is "false". Note also that we do not use σ' which is zero everywhere and non-existent at zero. Instead, we use s' which is always positive, so $\tilde{\delta}_j^l$ gives an indication of the direction and magnitude of a step up or down as a function of net_j^l in the error surface E.

However, as pointed out in [8], the value of the pseudo-gradient is not accurate enough, so gradient descent based training in DMNNs is considerably slow when compared with BP training in FNNs.

In order to alleviate this problem we propose an alternative to the pseudo-gradient training method procedure. To be more specific, we propose to solve the one-dimensional equation :

$$E(w_1, \ldots, w_{i-1}^0, w_i^0, w_{i+1}^0, \ldots, w_n^0) - E(w_1^0, \ldots, w_{i-1}^0, w_i^0, w_{i+1}^0, \ldots, w_n^0) = 0,$$

for w_1 keeping all other components of the weight vector in their constant values. Now, if $\hat{w}_1$ is the solution of the above equation, then the point defined by the vector $(\hat{w}_1, w_2^0, \ldots, w_n^0)$ possesses the same error function value with the point w^0, so it belongs to the same contour line of w^0. Assuming that the error function curves up from w^* in all directions, we can claim that any point which belongs to the line with endpoints w^0 and $(\hat{w}_1, w_2^0, \ldots, w_n^0)$ possesses smaller error function value than these endpoints. With this fact in mind we can now choose such a point, say, for example $w_1^1 = w_1^0 + \gamma\left(\hat{w}_1 - w_1^0\right)$, $\gamma \in (0,1)$, and solve the one-dimensional equation :

$$E(w_1^1, w_2, \ldots, w_{i-1}^0, w_i^0, w_{i+1}^0, \ldots, w_n^0) - E(w_1^1, w_2^0, \ldots, w_{i-1}^0, w_i^0, w_{i+1}^0, \ldots, w_n^0) = 0,$$

for w_2 keeping all other components in their constant values. If $\hat{w}_2$ is the solution of this equation then we can obtain a better approximation for this component by taking $w_2^1 = w_2^0 + \gamma\left(\hat{w}_2 - w_2^0\right)$, $\gamma \in (0,1)$.

Continuing in a similar way with the remaining components of the weight vector we obtain the new vector $w^1 = (w_1^1, \ldots, w_n^1)$ and replace the initial vector w^0 by w^1. The procedure can then be repeated to compute w^2 and so on until the final estimated point is computed according to a predetermined accuracy. So, in general we want to find the parameter $\hat{x}$ (a weight or threshold) that satisfies :

$$E(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x, x_{i+1}^k, \ldots, x_n^k) - E(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^k, x_{i+1}^k, \ldots, x_n^k) = 0,$$

by applying the modified bisection (see [12, 13]) in the interval (a_i, b_i) within accuracy d :

$$x_i^{p+1} = x_i^p + C\,\mathrm{sgn}\left(E(z^p) - E(z^0)\right)/2^{p+1}, \quad p = 0, 1, \ldots, \lceil \log_2((b_i - a_i)\,d^{-1}) \rceil,$$

where the notation $\lceil \cdot \rceil$ refers to the smallest integer not less than the real number quoted and $z^0 = (x_1^{k+1}, \ldots, x_{i-1}^{k+1}, a_i, x_{i+1}^k, \ldots, x_n^k)$, $z^p = (x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^p, x_{i+1}^k, \ldots, x_n^k)$, $C = \mathrm{sgn}E(z^0)(b_i - a_i)$, $a_i = \bar{x}_i^k - \frac{1}{2}\{1 + \mathrm{sgn}\widetilde{\partial_i E(\bar{x}^k)}\}h_i$, $b_i = a_i + h_i$. If an iteration of the algorithm fails we switch to the pseudo-gradient training method. So, the justification of the new procedure is based on the heuristic justification of the pseudo-gradient which can be found in any one of [8, 9, 10]. A formal justification of the proposed procedure in case of differentiable objective functions can be found in [11].

	BP			New method				
	MN	*STD*	*MNE*	*MN*	*STD*	*MNE*	*MAS*	*SAS*
a)	561	550.4	0.0396	40.6	4.2	0.0000008	239.6	54.12
b)	18121	3048.7	0.49	28.5	13.43	0.45	20673	9310.9

Table 1 Experimental results a) XOR, b) $\sin x \cos 2x$.

3 Experimental Results

Here we present and compare the behaviour of the new training method with the BP [4] and the pseudo–gradient training method [8] for the XOR problem and training an 1–10–1 network to approximate the function $\sin x \cos 2x$ (Table 1). In all problems $\gamma = 0.5, d = 10^{-10}, \bar{h} = 10$ and no pseudo–gradient subprocedure has been applied with the proposed method in order to get more fair evaluation. *MN* indicates the mean number of iterations; *STD* the standard deviation of iterations; *MNE* the mean value of the error; *MAS* the mean number of algebraic signs required for the bisection scheme and *SAS* the standard deviation of the required algebraic signs. The results are for 10 simulation runs, for the same initial weights; the maximum number of iterations was set to 2000, the weights were initialised in the interval $[-10, 10]$ and the step size for BP was set to the standard value 0.75. For the XOR the thresholds were set as follows: "*true*" = 0.8 and "*false*" = 0.2. Under the same conditions the pseudo–gradient training needed more than 2000 iterations to converge. The frequency with which the algorithm became trapped in local minima seems to be about the same as for BP for binary tasks. We also used the new method in training DMNN to learn smooth functions. One hidden layer of hard–limiting units and one output unit with linear activation function was used in all our experiments. We did not manage to train DMNNs using the pseudo–gradient training method due to oscillations, although various step sizes and different discrete activation functions have been tried. With the new algorithm and discrete activation functions such as 0.5 for "*true*" and -0.5 for "*false*" DMNNs were trained as fast as, and often faster than, BP trained FNNs until $E \leq 0.5$ (over 21 input/output cases). After this error bound, the convergence speed was reduced due to saturation problems.

However, it is worth noticing the difference in the behaviour between BP and the new method. Back–propagation trained FNNs exhibit a greater tendency to fit closely data with higher variation than data with low variation. On the other hand, although DMNNs do not produce smooth functions, they learn the general trend of the data values and therefore might be more useful than FNNs when there is noise in the data and the error goal can be set so high that the network does not have to fit all the target values perfectly. Situations like this usually occur in system identification and control (see [14]).

4 Conclusion and Further Improvements

This paper describes a new training method for DMNNs. The method does not directly perform gradient evaluations. Since it uses the modified one–dimensional bisection method it requires only that the algebraic signs of the function and gradient values be correct; so it can be applied to problems with imprecise function and gradient values. The method can also be used in training with block of network parameters, for example train the entire network, then the weights to the output layer and the thresholds of the hidden units, etc. We have tested such configurations and the results were very promising, providing faster training.

REFERENCES

[1] W. McCullough, W. H. Pitts, *A logical calculus of the ideas imminent in nervous activity*, Bulletin Mathematical Biophysics, Vol. 5 (1943), pp115–133.

[2] S. E. Hampson, D. J. Volper, *Representing and learning boolean functions of multivalued features*, IEEE Trans. Systems, Man & Cybernetics, Vol. 20 (1990), pp67–80.

[3] G. J. Gibson, F. N. Cowan, *On the decision regions of multi-layer perceptrons*, Proc. IEEE, Vol. 78 (1990), pp1590–1594.

[4] D. E. Rumelhart and J. L. McClelland eds., *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press (1986), pp318–362.

[5] B. Widrow, R. Winter, *Neural nets for adaptive filtering and adaptive pattern recognition*, IEEE Computer (March 1988), pp25–39.

[6] D. J. Tom, *Training binary node feed forward neural networks by back-propagation of error*, Electronics Letters, Vol. 26 (1990), pp1745–1746.

[7] E. M. Gorwin, A. M. Logar, W. J. B. Oldham, *An iterative method for training multilayer networks with threshold functions*, IEEE Trans. Neural Networks, Vol. 5 (1994), pp507–508.

[8] R. Goodman, Z. Zeng, *A learning algorithm for multi-layer perceptrons with hard-limiting threshold units*, in: Proc. IEEE Neural Networks for Signal Processing (1994), pp219–228.

[9] Z. Zeng, R. Goodman, P. Smyth, *Learning finite state machines with self-clustering recurrent networks*, Neural Computation, Vol. 5 (1993), pp976–990.

[10] Z. Zeng, R. Goodman, P. Smyth, *Discrete recurrent neural networks for grammatical inference*, IEEE Trans. Neural Networks, Vol. 5 (1994), pp320–330.

[11] M. N. Vrahatis, G. S. Androulakis, G. E. Manoussakis, *A new unconstrained optimization method for imprecise function and gradient values*, J. Mathematical Analysis & Applications, Vol. 197 (1996), pp586–607.

[12] M. N. Vrahatis, *Solving systems of non-linear equations using the non zero value of the topological degree*, ACM Trans. Math. Software, Vol. 14 (1988), pp312–329.

[13] M. N. Vrahatis, *CHABIS: A mathematical software package for locating and evaluating roots of systems of non-linear equations*, ACM Trans. Math. Software, Vol. 14 (1988), pp330–336.

[14] H. J. Sira-Ramirez, S. H. Zak, *The adaptation of perceptrons with applications to inverse dynamics identification of unknown dynamic systems*, IEEE Trans. Systems, Man & Cybernetics, Vol. 21 (1991), pp634–643.