AUTHOR(S): VISWANATHAN, A., MACLEOD, C., MAXWELL, G. and KALIDINDI, S.

TITLE: Training neural networks using Taguchi methods: overcoming interaction problems.

YEAR: 2005

Training Neural Networks Using Taguchi Methods: Overcoming Interaction Problems

Alagappan Viswanathan, Christopher MacLeod,
Grant Maxwell and Sashank Kalidindi

School of Engineering, The Robert Gordon University
Schoolhill, Aberdeen, UK

Abstract. Taguchi Methods may be used to train small Artificial Neural Networks very quickly in a variety of tasks. These include, importantly, Control Systems. Previous experimental work has shown that they could be successfully used to train single layer networks with no difficulty. However, interaction between layers precluded the successful reliable training of multi-layered networks. This paper describes a number of successful strategies which may be used to overcome this problem and demonstrates the ability of such networks to learn non-linear mappings.

1 Introduction

A number of papers have outlined the use of Taguchi and other Orthogonal arrays to train Artificial Neural Networks (ANNs). The idea was originated by C. MacLeod in 1994 [1] at the Robert Gordon University and implemented to practice by his student Geva Dror in an MSc project [2]. Another group at the JPL research centre in NASA also developed the same idea independently at around the same time [3].

The technique works by trying a series of different weight combinations on the network and then using Taguchi's techniques [4] to interpolate the best combination of these. A detailed description is not presented here due to space restrictions and the fact that the method is explained fully in several of the references [1, 3, 5]. The American team added to the basic technique by proposing an iterative approach [3].

The technique can operate very quickly to set network weights and it has been suggested that this could be used in "disaster control" networks [5] where the ANN takes over control from a damaged system (for example, an aircraft with compromised control surfaces).

The problem with the technique is that Taguchi Methods only operate well on variables which do not interact. In the case of a two layer ANN, the weights in the initial layer interact strongly with those of the second layer (that is, if you change the first layer weights, those in the second layer must also change if the error is to stay the same). This meant that, although it was occasionally possible to get a two layer network to train, more often than not, it did not.

Maxwell suggested some possible ways around this problem [5] but these did not work reliably in all cases. However, the methods outlined below do show

good results when used with two layer networks. They do this by treating the neurons, rather then the individual weights, as the basic units of the network.

2 Coding The State of Each Neuron

Since interaction between weights in different layers is the cause of the interaction described above, one possible way around this is to have each variable used in the Orthogonal Array (OA) correspond to a state of a particular neuron [6]. For example, an array with four levels could be used to code a two input neuron as shown in Fig. 1. The possible combinations of weights are shown in table 1. These then are used in the OA which represents the overall network.

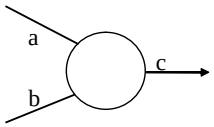


Fig. 1. Using levels to code neuron states.

Table 1. List of all weight combinations.

Level	Weight a	Weight b	Weight c
1	-1	-1	-1
2	-1	-1	+1
3	-1	+1	-1
4	-1	+1	+1
5	+1	-1	-1
6	+1	-1	+1
7	+1	+1	-1
8	+1	+1	+1

As with the original method [1], the weights are quantised. As noted in the references [5], although standard texts on the Taguchi method [4] give only simple arrays, it is possible to generate others using standard formulae [7, 8].

When this method is used, it does give a better error reduction than applying the standard method to a two-layer array. However, when compared against the full-factorial results for the same problem, although the error reduction is generally good, it is not as high as theoretically possible. The reason for this is neuron to neuron interaction replacing layer by layer interaction as a problem.

One can see this if one considers the structure of the experimental arrays. Consider a very simple example of the middle two experiments in a L4 array: 1 2 2 and 2 1 2. We can see that both these experiments correspond to the same

network (although the neurons are in a different order) as shown in Figs. 2(a) and 2(b). This means that, when we calculate the best state for a particular neuron, the system cannot differentiate between states 1 and 2 for a particular neuron as the table is symmetric.

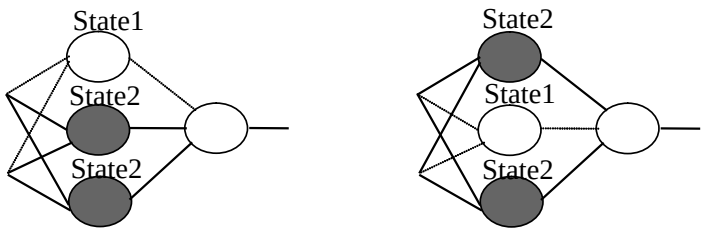


Fig. 2. Two different combinations of neuron weights which give the same result.

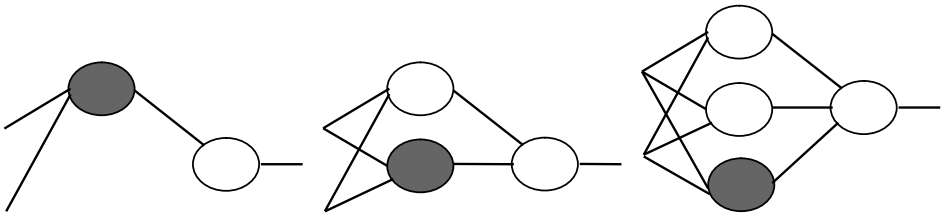
It is possible that this problem might be overcome by allocating different states to different levels in different neurons or by using interaction columns in the tables; however, this has not been tested.

The advantage of this approach is that reasonably good (although not optimum) results can be achieved. Its disadvantage is that large tables are required (as the size of the network increases) because the size of the table is proportional to the number of neurons.

3 Neuron By Neuron Training

A more successful technique is to train each neuron one by one. This does allow the error to fall to its lowest theoretical limit. It works as shown in Fig. 3. In network (a), the weights of the neuron are set using an orthogonal array in the usual way. These weights are then fixed and not altered during the rest of the training. Next, as in (b), a new neuron is added and its weights trained; the first neuron's output is used in the calculation of the error. Finally, the third neuron is added and the process is repeated, again using the first two neurons' outputs in the error calculation.

Apart from the guarantee of reaching a low error (within the limits that the quantised weights apply), this method also has the advantage that the orthogonal arrays used are small (because their size is proportional to the number of weights associated with an individual neuron, not the whole network). The disadvantage is that each succeeding neuron is refining the output and so the initial neuron has the greatest affect and each successive addition has less. In effect, the method acts rather like the iterative method discussed earlier [3]. Indeed, one can tailor the succeeding weights to refine the output in a similar way.



(a) First neuron's weights are trained

(b) Second neuron added and trained. First neuron used in error calculation

(c) Third neuron added and trained. Previous neurons used in calculation of error

Fig. 3. Neuron by Neuron training.

4 Power Series Neurons

Power series neurons are a refinement of perceptron types and are discussed in previous papers [9]. They allow a single neuron unit to fulfil any differential function as shown in Fig. 4. It was shown in earlier work [5, 9] that power series neurons can be trained using these methods.

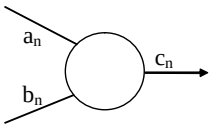


Fig. 4. A power series neuron

$$c = f((a_0.i_1 + b_0.i_2) + (a_1.i_1^2 + b_1.i_2^2) + (a_2.i_1^3 + b_2.i_2^3) + \ldots) \tag{1}$$

Where $f(x)$ is the activation function of x (typically a sigmoid), i_n^x is the x^{th} power of n^{th} input and a_n, b_n are the weights a and b associated with the $(n+1)^{th}$ power of the inputs.

Such a neuron can fulfil complex mathematical functions without having to resort to many layers. Although it is not capable in this form of separating discrete areas of input space, this is not necessary for fulfilling many functions required of control systems. It is possible to combine the power series and neuron-by-neuron approaches. Each additional power series neuron provides a new classifier.

5 Results

The training methods were shown to work by testing them on some non-linear mappings. These were a sigmoid, reverse sigmoid and gaussian. In this case the first method (coding the state of the neuron as a level) was used. The network consists of two inputs, one of which is held constant (at one unit); the other is varied as shown on the x axis. There are nine hidden layer and one output neuron. The array used is 64 rows of 8 levels coded as shown in Fig. 1 and generated using Owen's data [8]. The results are shown in Figs. 5, 6 and 7.

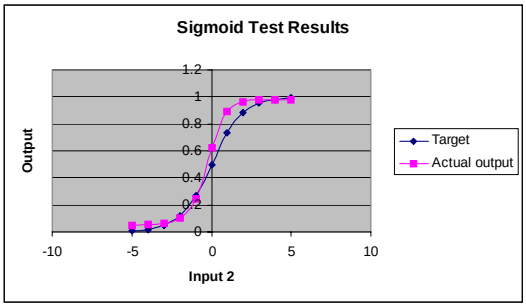


Fig. 5. Sigmoid Test

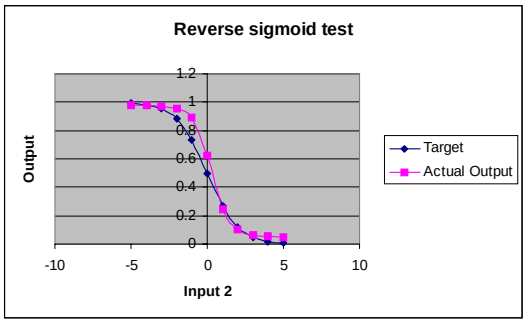


Fig. 6. Reverse Sigmoid Test

6 Conclusion

Using Orthogonal or Taguchi arrays to train neural networks is a promising technique. It allows relatively small networks to be trained very quickly and simply.

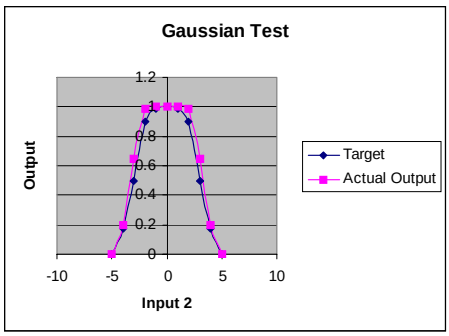


Fig. 7. Gaussian Test

This makes it an ideal technique for some specialised applications in control, particularly in the stabilisation of systems which may have undergone some change which makes their system model difficult or impossible to determine.

The method, however, suffers from the problem of interaction which means that it is difficult to apply to multi-layer networks. However, it is possible to overcome this problem. The three techniques discussed above are examples of how this may be done, the last two being particularly effective.

References

1. C. MacLeod, G. Dror and G. M. Maxwell, Training Artificial Neural Networks Using Taguchi Methods, *AI Review*, (13) 3, 177-184, Kluwer, 1999.
2. G. Dror, Training Neural Networks Using Taguchi Methods, MSc thesis, 1995, The Robert Gordon University.
3. A. Stoica, J. Blosiu, Neural Learning Using Orthogonal Arrays. In proceedings of the *International Symposium on Intelligent Systems*, Reggio (Italy), 418-423, IOS Press, Amsterdam, 1997.
4. R. K. Roy, *A Primer on the Taguchi Method*, Wiley, New York, 1990.
5. G. Maxwell and C. MacLeod, Using Taguchi Methods To Train Artificial Neural Networks In Pattern Recognition, Control And Evolutionary Applications. In proceedings of the *International Conference on Neural Information Processing*, (ICONIP 2002), vol 1, 301-305, Singapore, 2002.
6. A. Viswanathan, Using Orthogonal Arrays to Train Artificial Neural Networks, MPhil Thesis, forthcoming.
7. A. Dey, *Orthogonal Fractional Factorial Designs*, Wiley, New York, 1985.
8. A. Owen, Orthogonal Arrays for Computer Experiments, Integration and Visualization, Statistica Sinica 2, 439-452, 1992.
9. N. Capanni, C. MacLeod, G. Maxwell, An Approach to Evolvable Neural Functionality. In proceedings of the *International Conference on Artificial Neural Networks and Neural Information Processing*, (ICANN/ICONIP 2003), Istanbul (Turkey), 220-223.