NEURAL LEARNING ALGORITHMS : Some Empirical Trials

Richard Forsyth,
Dept. of Psychology
University of Nottingham
Nottingham NG7 2RD, UK.

Abstract: The recent revival of Connectionism has led to an upsurge of interest in trainable pattern associators and pattern classifiers of many types. However, one training method currently dominates the field -- the back propagation algorithm. This method is crowding out other neural learning algorithms and other inductive techniques. The present paper reports some empirical trials comparing seven different neural learning algorithms (including two versions of back propagation) on four test problems. Though limited in scope the present study does shed light on the performance of a variety of learning techniques, compared under relatively uniform conditions. The results cast some doubt on the status of back propagation as an 'industrial strength' learning algorithm. It appears to scale up rather poorly; and on two pattern recognition tasks it gave a higher error rate than a commonly used statistical technique. These results suggest that the neurocomputing community as a whole may be in danger of becoming fixated at a local optimum, just like some of its algorithms.

Keywords:
Machine Learning, Neural Computing, Back Propagation, Genetic Algorithms, Simulated Annealing, Distributed Memory.

1. Introduction

The recent resurgence of interest in Neural Computing, also known as Connectionism or Parallel Distributed Processing, has led to a proliferation of computing systems based (sometimes rather loosely) on the operation of the nervous system. There are dozens, possibly hundreds, of different neuro-computing models, each one with its own particular strengths and weaknesses; yet one in particular so dominates the field as to have become almost a 'connectionist cliché'. Neural nets that exemplify this cliché have the following characteristics in common:

(1) they have 1 or 2 hidden layers;
(2) processing units are fully connected between layers;
(3) processing units are not connected within layers;
(4) a sigmoidal transfer function is used throughout;
(5) they work in feedforward mode only (i.e. such nets are not recurrent);
(6) they are trained by adjusting connection-weights using the Backward Error Propagation scheme.
(See: Werbos, 1974; Parker, 1982; Rumelhart et al., 1986.)

In this paper I will refer to networks exhibiting these six characteristics as 'back-propagation' systems, since it is the use of a particular training algorithm, Backward Error Propagation, which determines the other five features.

Back propagation is well on its way to becoming an industry standard. Yet it suffers from a number of well known problems including poor scalability (Tesauro, 1987) and 'network paralysis' (Wasserman, 1989). Nor is there any shortage of reports describing alternative neural computing models which outperform back propagation in speed, accuracy or both.

For example, Aleksander (1989) describes a network composed of 3 Probabilistic Logic Nodes (PLNs) trained by his PLN learning algorithm that reached a solution to the 4-bit parity problem on average 3000 time more quickly than a network trained by back propagation. Huang & Lippmann (1987) found that a feature-map classifier, employing a method based on that of Kohonen (1984), learned to discriminate between ten different vowel sounds in 50 presentations while a 2-layer back-propagation system needed 50000 presentations to attain comparable accuracy. Hampson & Volper (1987) discuss a learning algorithm that learned to solve several Boolean mapping problems an order of magnitude faster than a multi-layer Perceptron trained by back propagation. Finally, Shepanski (1988) compared a network 'trained' by standard numerical optimization techniques against a back-propagation network with 2 hidden layers on a problem involving the reconstruction of a noisy input signal. Although his method requires that all training cases be in memory at once, and hence cannot be used incrementally, it proved 117 times faster and 39 times more accurate than back propagation on a particular signal detection task.

Yet, despite studies such as those cited above, back-propagation systems abound; and the method occupies a position of prominence in the textbooks, seminars and software that tend to introduce newcomers to the field.

2. Object of the Exercise

The aim of the present study is to shed some more light on the status of back propagation as a learning algorithm by performing some empirical trials. Accordingly, seven different neural learning programs were written by the author in the same programming language and applied to four different test problems. This format is conceptually simple; nevertheless, it is hoped that it will serve a useful purpose by comparing a larger number of methods than is usual on a relatively 'level playing field'.

3. Seven Neural Learning Algorithms

Six basic algorithms were chosen for testing -- one in two variants -- giving seven distinct methods. They will be described in order of increasing complexity. All except the last are weight-adjustment methods, applicable to feedforward nets.

3.1 Random Search

Method 1 (referred to hereafter as RANDNET) employs simple Monte Carlo Search. It was included merely to establish a baseline performance target.

At each step the RANDNET program generates a complete weight-set completely anew, runs the network over the training data using that weight-set to compute the average squared error and saves the weights if they happen to yield the lowest error score so far. Though nobody has, to my knowledge, proposed this as a serious neural learning algorithm its performance gives a useful index of the difficulty of the problem -- i.e. of the proportion of weight-sets in the search space that represent solutions.

3.2 Greedy Search

Method 2 (GREEDNET) is a 'greedy' algorithm (Aho et al., 1983). On each main cycle it mutates the current weights, tests whether the revised weights are better than the current weights and, if so, replaces the current weights with the revised ones.

Viewed as a hill-climbing method, GREEDNET takes small random steps from the current location in weight space, but only accepts them if they cause downhill movement on the error surface. A similar method has been described by Wasserman (1989), though his mutation procedure only alters a single weight at a time, whereas GREEDNET mutates all weights at once, by adding pseudo-random numbers in the range -0.125 to 0.125, uniformly distributed.

3.3 Simulated Annealing

Method 3 (ANNA) is the first of the 'serious' neural learning algorithms. The program uses simulated annealing which is a widely used combinatorial optimization technique and has been proposed as the basis for neural-net training algorithms (Ackley, 1987; Wasserman, 1989).

Simulated annealing was first reported by Metropolis et al. (1953). It is a general-purpose heuristic which can be applied to a range of problems. It is based on annealing, a process of slowly cooling molten metal. At high temperatures, the molecules of a liquid move freely. As the liquid cools down, thermal mobility is lost. Normally the atoms line themselves up to form a pure crystal that is perfectly orderly over distances billions of times the size of the individual atoms. This represents the minimum energy state for the solid. (See also Kirkpatrick et al., 1983.)

Nature nearly always finds this minimum energy state provided the material is cooled sufficiently slowly. The algorithm uses annealing as a model to find the system configuration with minimal cost. It starts off with a feasible but non-optimal solution and keeps making minor changes to it. The essential point is that all changes which improve the potential solution are retained but also some changes that make it slightly worse. These latter correspond to random fluctuations into higher energy states, which are unlikely but not impossible. In fact, the probability of a deleterious change being accepted is

$$\exp(-D / (k*T))$$

where D is the difference in cost (positive D indicating a worse solution), k is a scaling parameter, corresponding to Boltzmann's constant, and T is the temperature variable.

To use simulated annealing to optimize a system the following elements are needed:

(1) a description of possible system configurations;
(2) a generator of random changes to the current configuration;
(3) an objective function whose minimization is the goal of the procedure;
(4) a control parameter T which is the analogue of temperature and which is gradually lowered.

ANNA (Adaptive Neural Net Annealing) implements these requirements for weight-adjustment in a feedforward net. It is based on the procedure listed by Press et al. (1986), who used it to seek minimal cost tours in the Travelling Salesman Problem. Here it is used to find weight-sets that give rise to low error rates. The main differences between their program and the present one are (a) that in ANNA initial temperature is computed as 1% of the initial cost and (b) that the temperature parameter is lowered only when the last 20 trials have failed to produce a new least-cost weight-set. The former modification saves the user from having to pick a suitable initial temperature; the latter allows the system to remain at a given temperature level as long as it appears to be making progress at that level.

Note that this program does **not** implement a "Boltzmann Machine" (Ackley et al., 1985). The Boltzmann Machine uses similar ideas in a very different way.

3.4 Back Propagation

Method 4 (BP/V) I call the "plain vanilla" version of back propagation. It is the classic algorithm as described in Rumelhart & McClelland (1986), with weight updating after each pattern.

It is unlikely that anyone is using exactly this version of the algorithm today, except for pedagogic purposes, but it can fairly claim to be the prototype or progenitor of all the many back-propagation systems in widespread use, and is included here for that reason. It has the great merit of not confronting the would-be user with any free parameters to be tuned.

3.5 Back Propagation with Momentum

Method 5 (BP+M) is a more highly tuned version of the back propagation algorithm. It is actually the same program as method 4 with different parameter values. For all runs reported here, the learning coefficient was set to 0.5 and the momentum term to 0.75. These values were chosen after a scan through the literature and some preliminary runs of the system with one hidden layer on the 4-bit parity problem.

There is no unanimity in the literature about the choice of these parameters, as shown by Table 1, even though correct choice can make the difference between a network that learns fast and one that does not learn at all. The present settings are within the range of typically reported values, though they are unlikely to be optimal. (It is possible that correct choice of learning rate and momentum is highly problem-dependent.)

[Table 1 -- Reported Parameter Settings (BP).]

Learning Coefficient	**Momentum Term**	**Author(s)**
0.3	0.7	(Lippmann, 1987)
1.0	0.9	(CSS Inc., 1988)
-	0.9	(Sejnowski & Rosenberg, 1987)
0.25	0.9	Rumelhart & McClelland, 1986)

3.6 Genetic Search

Method 6 (Genetic Evolutionary Network Evaluator) uses a genetic algorithm for weight-adjustment. Many kinds of genetic algorithm have been devised with a variety of applications (Holland, 1975; Smith, 1980; Forsyth, 1981; Rechenberg, 1989; Goldberg, 1989) -- including at least one used to optimize the weights of a neural network (Whitely, 1989).

In general, a search or optimization problem is suited by a genetic, or evolutionary, approach if the following conditions hold:

(1) candidate solutions can be represented by a uniform encoding;
(2) candidate solutions may be rated or at least ranked by a *fitness function*;
(3) new potential solutions can be produced by slicing and recombining portions of existing candidate solutions;
(4) the search space is large.

In GENE, candidate solutions are held as byte strings for the purposes of mutation and recombination, and decoded into an array of floating-point numbers for evaluation by running the network. (Thus each weight is held with a precision of only 8 bits.) The fitness function is simply the mean squared error of the network when run over the training dataset.

The GENE program implements a version of the genetic algorithm due to Ackley (1987) which he calls IGS/U -- Iterated Genetic Search with Uniform crossover. The only difference is that GENE uses a fixed mutation rate (probability 0.075 per byte) while IGS/U uses a decreasing mutation rate (by analogy with simulated annealing).

(Note that mutation in the GENE program, which consists of flipping one or more randomly chosen bits in the byte-string representation of the weights, is not identical to mutation in the programs ANNA and GREEDNET, where small uniformly distributed random numbers are added to weights that are held as floating-point values.)

3.7 Cerebellar Model Articulated Controller

Method 7 (CMAC) is the only one of the seven learning methods which does not search weight space. Instead it implements the CMAC memory model of James Albus (1981) which utilizes a form of distributed memory. Though thoroughly neurological in inspiration, being based on a theory (Albus, 1971) of cerebellar function, CMAC does not use nets of neuron-like processing units connected by weighted links. Rather it embodies certain aspects of the functionality of the Cerebellum -- that part of the brain concerned with balance and skilled motor coordination -- by means of a sophisticated table look-up scheme.

The crucial difference between CMAC and ordinary table look-up is that CMAC calculates a *set* of addresses from each input rather than a single address. In the present program each input vector gives rise to a set of 25 addresses computed using 25 different hashing functions (strictly, one hashing function parameterized in 25 ways). The responsibility for the system's response is shared by 25 memory cells.

The essential idea is the similar inputs give rise to overlapping sets of memory addresses and thus to similar responses. The weights in the memory table are revised during training by a variant of the Widrow-Hoff error-correction method (Widrow & Hoff, 1960).

4. The Four Test Problems

The first two test problems call upon the systems' ability to learn logical associations, and were mainly intended to test speed and accuracy of learning. The second two problems require the systems to act as pattern recognizers and were mainly intended to test the ability to generalize.

4.1 Exclusive OR

Problem number 1 (EXOR) is the ubiquitous Exclusive-Or problem, identified by Minsky & Papert (1969) as the simplest Boolean function that a one-layer Perceptron (Rosenblatt, 1962) cannot solve. It has perhaps received more than its fair share of attention already, but it is included none the less as a link between the present work and earlier studies.

In the EXOR problem there are two binary inputs and one output line, and the programs must learn the relationship between input and output. Since there are only four possible examples, the complete training set is listed in Table 2.

[Table 2 -- EXOR.]

Inputs		Outputs
0	0	0
0	1	1
1	0	1
1	1	0

All networks were configured for this problem with one hidden layer of two nodes (except for CMAC, which does not use weighted links).

4.2 Four-bit Parity

The second test problem, 4-bit parity, extends the EXOR problem from two input lines to four. Here the programs must learn to respond with a 1 when an even number of inputs are on (1) and zero when an odd number are on. In this case there are 16 possibilities, so each pass through the training set implies 16 training presentations.

For this problem (PARITY4) all the nodes were configured with two hidden layers of four nodes each (except for CMAC, which does not use weighted links).

4.3 Zoological Classification

The third data-set (ZOOBASE) requires the system to learn to discriminate between different classes of animals. This data-set contains details of 101 animal species. Each species is described in terms of 17 features, such as the number of legs it has or whether it gives milk to its young. There are 7 output lines, all of which are off (0) except one -- indicating the zoological class of the species concerned.

1. Mammal
2. Bird
3. Reptile
4. Fish

5. Amphibian
6. Insect
7. Other zoological family

To test the systems' capacity to generalize from seen to unseen cases, the data was randomly split into two subsets -- a training set of 57 cases and a test set containing the remaining 44 examples.

The weighted-link nets were trained on this data using one hidden layer of 17 nodes.

4.4 Chocolate Silhouettes

The fourth and last test (CHOX) requires the nets to perform a simplified industrial inspection task. This data (obtained from the Turing Institute in Glasgow) comes from a larger database of images previously described by Shepherd (1983). The CHOX file contains 80 records in all, each with nine measurements obtained from silhouette photographs of different chocolates. These attributes describe the geometric features of the chocolates as seen by a simplified robotic vision system, with a resolution of 96x96 pixels.

The objective is to train a system to assign the image of each chocolate to its correct class, of which there are eight. The shape descriptors used were as follows. (Raw feature scores were multiplied by 100 and rounded to the nearest integer.)

area,	area of centred image
prornd,	area / area of circle with same perim.
circularity,	a measure of circularity
boxness,	a measure of rectangularity
box2ness,	area / area of computed rectangle
asprat,	aspect ratio
as2rat,	max.length / max.width
perimeter,	length of perimeter of normalized image

The 80 cases were randomly divided into two subsets, 44 training examples and 36 test cases. There were eight output lines of which all were zero except one, the line corresponding to the chocolate type that produced the image. The nets (apart from CMAC, see above) were configured with nine input lines, eight outputs and one hidden layer of nine units for all reported trials.

Of the four tests employed, this is the closest to a real application.

5. Test Results

The results are presented in two sections. Section 5.1 deals with the two noise-free problems, used to assess speed and accuracy of learning. Section 5.2 is intended to assess the systems' capacities for generalization.

5.1 Learning Speed and Accuracy

All seven methods were run five times on the EXOR problem, each time taking 2000 passes over the training examples. Table 3 shows the median error score (i.e. the 3rd-best of five runs) attained after 2000 passes over the training data by each method. The error score is the squared deviation from the true va1ue (0 or 1) of the system's output averaged over the four possible inputs.

[Table 3 -- Error Scores on EXOR.]

Method	Error-Score
CMAC	0.0000
BP+M	0.0004
GREEDNET	0.0008

GENE	0.0123
ANNA	0.0250
BP/V	0.1263
RANDNET	0.1414

The methods have been placed in rank order. This same data is presented as a bar-chart in Figure 1, which illustrates that there were basically two methods that failed (BP/V and RANDNET) and five that worked (the rest).

The best score was obtained by CMAC, which always converged to a mean error score of less than 0.0001 within 250 passes. This is hardly surprising, with hindsight, given that CMAC is a table look-up system and the simplest way to solve the EXOR problem (if it really were a problem) would be to look up the answer in a table.

Really all these figures show is that the EXOR problem is not challenging enough to discriminate between different learning methods. The only finding to emerge is the divergence between BP/V and BP+M, confirming that back propagation is very sensitive to correct choice of learning-rate and momentum parameters.

The 4-bit parity problem, however, is a more realistic test, and here the results start to show clear differences between the various methods. Table 4 lists the mean squared errors after 2000 passes (again the median of 5 runs) for all seven methods.

[Table 4 -- Error Scores on PARITY4.]

Method	Error Score
CMAC	0.0001
GENE	0.1516
ANNA	0.2119
RANDNET	0.2379
GREEDNET	0.2438
BP+M	0.2500
BP/V	0.2500

Here the superiority of the rote-memory method (CMAC) over the others is obvious. In fact, CMAC always converged to within 0.0025 of the correct answer on all 16 inputs within 350 passes. This data is presented graphically in Figure 2.

This should not be construed as saying that back propagation (or the other weight-space search methods) cannot solve the 4-bit parity problem -- given time and a judicious set of parameter values. Nor should it be concluded that CMAC is invariably the method of choice for neural learning. It is safe to say, however, that back propagation scales up rather poorly from a simple problem (EXOR) to a more complex version of the same task (PARITY4). Indeed, neither of the back-propagation methods ever got below a mean squared error of 0.25, while RANDNET, which is a pure Monte Carlo search, did so on every one of its five runs; and a score of 0.25 represents complete ignorance: it results from giving the same output (0.5) to each of the 16 inputs.

The best of the weight-space search methods was GENE, a genetic algorithm, which illustrates the robustness of this particular search technique, although the weights found by GENE never gave less than one mistake (defined as a squared error score of more than 0.25) out of 16 cases in post-testing.

The apparent superiority of GENE (a genetic algorithm) over ANNA (simulated annealing) is also interesting. At the end of all five runs the error scores obtained by GENE were lower than the error scores obtained by ANNA on all its five runs. This suggests that it is important (at least in domains with a convoluted search space) to keep a population of candidate solutions in play, rather than just one.

Loosely speaking, GENE conducts N searches in parallel, where N is the population size, while ANNA only conducts one. On difficult search spaces -- i.e. those with irregular gullies, hills and ravines -- this can often be beneficial.

Another contrast is between the direct methods (BP+M and BP/V) and the stochastic methods (ANNA and GENE). In this problem the stochastic methods fared better, though neither could be called satisfactory.

It is also worth noting that the back-propagation programs were the slowest of all on this problem (PARITY4). Table 5 gives the mean runtime for 2000 passes, rounded to the nearest second, taken by each method on a 386-based computer.

[Table 5 -- Average Runtimes.]

Method	Runtime	Ratio
RANDNET	661	1.000
GREEDNET	680	1.029
ANNA	688	1.041
GENE	779	1.179
CMAC	1511	2.286
BP/V	2261	3.421
BP+M	2271	3.436

This comparison is rather unfair on CMAC, since it always converged within 350 passes and the timings are given for 2000 passes. If time-to-convergence had been the criterion it would easily have been the fastest (at around 280 seconds). Even so, it proved quicker than back propagation.

Of course, object of the exercise was not to optimize each program for speed, so these figures should be treated with caution. Nevertheless, they portray back propagation in a most unflattering light, and tend to support the contention that training a network by this method is a slow process.

5.2 Generalization Capability
It is one thing to adapt to a training-set containing noise-free instances of a simple logical relationship; it is something else to perform the act of induction -- to learn enough about the characteristics of a set of exemplars to respond successfully to unseen instances of the same sort, especially when, as is frequently the case, the training sample may be imperfect or incomplete.

To assess this aspect of performance, the two methods that did best on the harder of the two logical problems (ANNA and GENE) as well as the better of the back-propagation methods (BP+M) were applied to two more realistic data-sets. In addition, to give a slightly broader perspective, a program was written that implements a well-established statistical classification method, the Nearest-Neighbour technique (Fix & Hodges, 1951) and applied to the same data.

The Nearest-Neighbour method classifies an input vector by scanning through the training database and computing the Euclidean distance in feature space between the input example and all the training cases. The new input is assigned to the same class as the training example that is closest to it, i.e. has the minimal sum of squared differences over all attributes. The nearest-neighbour program used here is based on that listed by James (1985), with modifications to accept data in the same format as the neural-net programs. (Note: when being re-run on the training file, this program computes the distance of each case to all **other** exemplars, excluding the current case itself.)

For the comparisons reported below, success rate (percentage of cases correctly classified) is used as the primary performance measure, rather than the mean squared error, because the systems are being used as classifiers. To force each system to give an unambiguous classification, the output line with the highest value was always picked as the system's decision.

Tables 6 shows the results on the ZOOBASE data after 200 passes over the training data, for the better of two replications. (The default strategy involves simply picking the commonest category in the data-set concerned. It shows the proportion of correct choices to be expected by chance.)

[Table 6 -- Success Rates on ZOOBASE.]

Method	Training-Set%	Test-Set%
CMAC	100	70
BP+M	100	82
ANNA	60	55
GENE	54	39
Nearest-Neighbour	95	89
Default Strategy	46	34

Table 7 presents the results on the CHOX data, again after 200 passes over the training set, for the better of two repetitions.

[Table 7 -- Success Rates on CHOX.]

Method	Training-Set%	Test-Set%
CMAC	100	86
BP+M	80	72
ANNA	39	22
GENE	32	28
Nearest-Neighbour	89	94
Default Strategy	18	17

Here back propagation redeemed itself somewhat, at least in that BP+M worked far better than ANNA and GENE. BP+M converged to a solution with the ZOOBASE training data inside 200 passes and came reasonably near convergence with the CHOX data, while ANNA and GENE hardly got started.

CMAC and BP+M showed some signs of *overfitting* the training data. CMAC, at least the present version, would appear to have rather too many degrees of freedom. It managed to fit the training data perfectly both times, and did show some ability to generalize; but it was not as accurate on unseen data as the nearest-neighbour method. BP+M adapted nearly as well as CMAC to the training data and its performance profile on the test data was very similar to that of CMAC. Once again BP+M was the slowest of the methods tested.

6. Some Conclusions

This study is limited in aims and scope. It can only be regarded as a pilot for further, more extensive, empirical investigations. Nevertheless, it does cast light on the efficiency of a variety of current neural learning methods.

The first thing to say is that (unfortunately but also unsurprisingly) no overall 'champion' method emerges from these trials. Nevertheless some trends may be discerned in the results, and some general remarks about the place of neural methods within the field of machine learning would appear to be justified.

6.1 Individual Methods

Back propagation, it seems fair to say, is very good when it works well, but when it works badly it is awful. In other words, it is a brittle technique. It tends to scale up poorly, and can only be used with confidence by someone who is prepared to put considerable effort into getting the network architecture right and tuning the learning parameters. Also it is very slow.

The genetic algorithm, in this context, would appear to be robust but slow. That is to say, it was the best of the weight-space search methods on the most difficult problem (PARITY4) but took much longer to learn than back propagation on the easier problems (ZOOBASE and CHOX).

Simulated annealing, it would appear, falls between back propagation and the genetic search. It does slightly better on the easy problems than the genetic search but slightly worse on the hard one. Conversely it is worse than back propagation on the easier problems but more robust on the hard one. It may well deserve more attention than it has received as a viable compromise between speed and robustness. (And, like the genetic algorithm but unlike back propagation, simulated annealing can be used to optimize non-numeric knowledge-structures, such as symbolic descriptions.)

CMAC was the only method that always fitted the training data perfectly; and it did show an ability to generalize to unseen cases comparable with that of back propagation. However the version used here was not as robust in the face of noise as the nearest-neighbour classification technique.

(There is little point in dwelling on the other methods.)

6.2 General Comments

A few more general observations are also worth making.

Firstly, in view of the optimism displayed by some members of the neuro-computing fraternity, the fact that all the neural methods were beaten by a commonplace statistical technique can only be described as embarrassing. Nearest-neighbour classification was included here to serve roughly the same function as a *placebo* in a clinical trial. Yet it outperformed the more sophisticated algorithms. The nearest-neighbour method has well-known shortcomings (e.g. need for memory to store all training instances and vulnerability to disparities of scaling between features) but if as much effort were devoted to refining this method, and to devising massively parallel hardware for implementing it, as to building specialized 'neuronal' devices, we might well end up with more pattern-association machines than by following the route we appear to have chosen. At the very least, this is a sobering thought.

Secondly, these results do seem to bear out the contention that a disproportionate amount of effort is going into systems based on one particular learning algorithm. On the face of it, back propagation simply does not deserve its position of dominance. Its historical importance cannot be gainsaid; but it is a mistake to regard back propagation as *the* neural learning algorithm.

Finally, it is appropriate to sound a warning against becoming fixated upon a single neural architecture (known to be physiologically unrealistic). It is not true that all neural models have to be composed of processing elements that sum their inputs, apply a uniform transfer function to this sum and then pass on the resulting activation value, via weighted links, to other nodes of the same type -- still less is it necessary that such nodes be organized into rigidly demarcated layers.

The real problem with back propagation is not that it sometimes fails to learn in a reasonable time. A far more serious problem is that it narrows the vision of the entire neuro-computing community. It only works with nets of a certain type; consequently only nets of that type are seriously studied. Systems of a radically different nature, such as those of Albus (1981), Reilly et al. (1982) or Aleksander (1987), tend to receive relatively little attention as a result. Yet the performance of CMAC -- which is inspired by neurological ideas yet breaks away altogether from the notion of nodes connected by weighted links -- was in the current study more impressive than any of the 'conventional' neural models tested.

Perhaps the neuro-computing community as a whole is in danger of becoming trapped at a *local optimum*, like some of its algorithms?

7. Acknowledgements

I would like to thank Dr Simon Jones of Loughborough University and Prof. Igor Aleksander of Imperial College, London, for comments on an earlier draft of this paper. (They should not, of, course, be blamed for any faults that remain.) Thanks are also due to the Turing Institute in Glasgow for kindly supplying the CHOX data-set.

Postscript: the datasets described herein are available on request from the author (on PC-compatible diskettes only). Please accompany any requests for copies with a 5.25" floppy disc.

References

Ackley, D.H. (1987) -- An empirical study of bit vector function optimization. In Davis, L. (ed.) Genetic algorithms & simulated annealing: Pitman, London.

Ackley, D.H., Hinton, G.E. & Sejnowski, T.J. (1985) -- A learning algorithm for Boltzmann machines: Cognitive Science, 9, pp 147-169.

Aho, A.V., Hopcroft, J.E. & Ullman, J.D. (1983) -- Data structures & algorithms: Addison Wesley, Reading Mass.

Albus, J.S. (1971) -- A theory of cerebellar function: Mathematical Biosciences, 10, pp 25-61.

Albus, J.S. (1981) -- Brains, behavior & robotics: McGraw-Hi1l, Peterboro, NH.

Aleksander, I. (1987) -- Adaptive vision systems and Boltzmann machines: a rapprochement: Pattern Recognition Letters, 6, pp 113-120.

Aleksander, I. (1989) --The logic of connectionist systems: North Oxford Academic Publishers, London.

California Scientific Software (1988) -- Brainmaker users guide & reference manual: CSS Inc., CA.

Fix, E. & Hodges, J.L. (1951) -- Discriminatory Analysis, nonparametric classification: USAF School of Aviation Medicine, Report no. 4.

Forsyth, R.S. (1981) -- BEAGLE, a Darwinian approach to pattern recognition: Kybernetes, 10, pp 159-166.

Goldberg, D.E. (1989) -- Genetic algorithms in search, optimization & machine learning: Addison Wesley, Reading, Mass.

Hampson, S.E. & Volper, D.J. (1987) -- Disjunctive models of Boolean category learning: Biological Cybernetics, 56, pp 121-137.

Holland, J.H. (1975) -- Adaptation in natural and artificial systems: Univ. Michigan Press, Ann Arbor.

Huang, W.Y. & Lippmann, R.P. (1987) -- Neural net & traditional classifiers: Conf. on Neural Information

Processing Systems, IEEE, Nov. 1987.

James, M. (1985) -- Classification algorithms: Wm. Collins Sons & Co. Ltd., London.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983) -- Optimization by Simulated Annealing: Science, 220, pp 671-680.

Kohonen, T. (1984) -- Self-organization & associative memory: Springer-Verlag, Berlin.

Lippmann, R.P. (1987) -- An introduction to computing with neural nets: IEEE ASSP magazine, April 1987, pp 4-22.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. & Te1ler, E. (1953) -- Equations of state calculations by fast computing machines: J. Chemistry & Physics. 21, pp 1087-1091.

Minsky, M. & Papert, S. (1969) -- Perceptrons: an introduction to computational geometry: MIT Press, Cambridge, Mass.

Parker, D.B. (1982) -- Learning-logic: Invention report S81-64, file 1, Office of technology licensing, Stanford Univ.

Press, W., Flannery, B., Teukolsky, S. & Vetterling, W. (1986) -- Numerical recipes: Cambridge University Press, Cambridge.

Rechenberg, I. (1989) -- Artificial evolution & artificial intelligence. In Forsyth, R.S. (ed.) Machine learning: principles & techniques: Chapman & Hall, London.

Reilly, D.L., Cooper. L.N., Elbaum, C. (1982) -- A neural model of category learning: Biological Cybernetics, 45, pp 35-41.

Rosenblatt, F. (1962) -- Principles of neurodynamics: Spartan Books, NY.

Rumelhart, D.E. & McClelland, J.L. (1985) -- Parallel distributed processing, vol. 1: MIT Press, Cambridge, Mass.

Sejnowski, T.J. & Rosenberg, C.R. (1987) -- Parallel networks that learn to pronounce English text: Complex Systems, 1, pp 145-168.

Shepanski, J.F. (1988) -- Fast learning in artificial neural systems: Proc. 4th AI-West Conf., Long Beach, Tower Conf. Management Publishing, Glen Ellyn, Illinois.

Shepherd, B.A. (1983) -- An appraisal of a decision tree approach to image classification: Proc. 8th IJCAI, Karlsruhe, pp 473-475, Morgan Kaufmann publishers, Los Altos, CA.

Smith, S.F. (1980) -- A learning system based on genetic adaptive algorithms: Doctoral dissertation, Univ. Pittsburgh, PA.

Tesauro, G. (1987) -- Scaling relationships in back propagation learning; dependence on training set size: Complex Systems, 1, pp 367-372.

Wasserman, P. D. (1989) -- Neural computing: theory & practice: Van Nostrand Reinhold, New York.

Werbos, P.J. (1974) -- Beyond regression: new tools for prediction & analysis in the behavioral sciences Masters thesis, Harvard Univ., Mass.

Whitley, D. (1989) -- Applying genetic algorithms to neural network learning. In Cohn, G. (ed.) Proc. 7th Conf. of SSAISB: Pitman, London.

Widrow B., & Hoff, M.E. (1960) -- Adaptive switching circuits: IRE Wescon Convention Record, 4, pp 96-104, Inst. of Radio Engineers, New York.