Complex Systems **2** (1988) 671–704

Hard Learning the Easy Way: Backpropagation with Deformation

Frank J. Śmieja
Gareth D. Richards
Department of Physics, University of Edinburgh, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom

Abstract. The backpropagation algorithm for feed-forward layered neural networks is applied to a problem domain of variable difficulty. The algorithm in its basic form is shown to be very sensitive to step-size and momentum parameters as problem difficulty is increased. To counter this we suggest a way of changing them during learning for a faster and more stable gradient descent. A technique for the deformation of the error surface is introduced as a way of using the algorithm to learn hard problem domains by gradually changing the shape of the error surface from a gentle to the final craggy form. This deformation procedure is applied to a second problem domain and is shown to improve the learning performance by gradually increasing the difficulty of the problem domain so that the net may build upon past experience, rather than being subjected to a complicated set of associations from the start.

1. Introduction

The need for a neural network to employ non-linear representations of the tasks it is required to learn, calls for the use of hidden units (units whose states are not clamped in any part of the learning process), and an algorithm capable of adjusting the system's links so that links to and from hidden units may be updated after each question/answer (picture) training session such that the system may eventually associate each question with its correct answer. Such an algorithm has recently been suggested [8], and employs gradient descent to minimize a cost function, defined by the discrepancy between actual network output and desired output. The performance of this algorithm, and modifications made to it, are described in this paper with reference to an easily defined task which the network was required to perform. We investigate the performance of the basic algorithm during the learning, noting the advantages of adding hidden units. We also investigate general features of learning using maps in weight-space, which suggest ways for improving the descent by changing the step-size during learning so that momentum can safely be used.

Exploration of the characteristics of the error surface in weight space also prompt the introduction of a technique enabling the network to learn hard problem domains, by descending an error surface which is progressively deformed into that of the hard problem domain from a set of simpler problem domains. The simpler domains comprise easier parts of the hard domain. This technique is in a similar vein to the process of gradual acquisition of more refined skills in humans, starting from less specific basic skills. Finally, we demonstrate another use of the deformation technique used in this paper, in teaching a network another hard problem domain which involves descending a treacherous error surface. After this work was completed we noticed some similar work developed independently by Wieland and Leighton [10]. We hope that further work in this area will help determine the class of problems on which the deformation technique can be used.

The network used in this paper was layered feed-forward with input, output and hidden layers. Connections were only allowed between adjacent layers, and there were no connections within layers. The potential at unit i has the form

$$\phi_{ip} = \sum_j w_{ij} g_{jp} + U_i \tag{1.1}$$

where w_{ij} is the weight from unit j to unit i, g_{jp} is the output state of unit j for picture presentation p. U_i represents the "threshold value" for unit i. It can be updated during the learning cycle simply by treating it as a weight from a unit which is permanently "switched on" (i.e. its output state does not change during each picture presentation, and has a finite value); this unit is labelled by the index $j = 0$, thus $U_i = w_{i0} g_0$.

The response function of the units, $g_i(\phi_i)$, was taken to be

$$g_i(\phi_i) = 2f_i - 1 \tag{1.2}$$

with

$$f_i = v_i \tag{1.3}$$

for input units, and

$$f_i = \frac{1}{1 + e^{-\phi_i}} \tag{1.4}$$

for all other units, where v_i is an input picture, and the output was taken to be

$$o_i = \frac{g_i^{(out)} + 1}{2} \quad (= f_i), \tag{1.5}$$

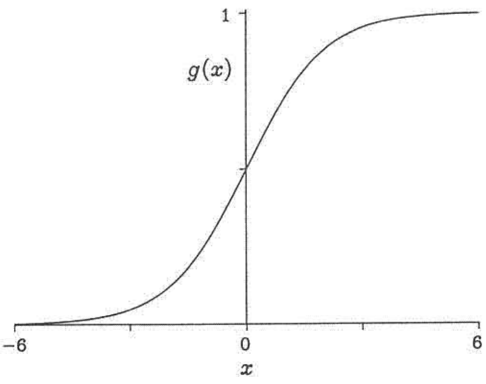


Figure 1: The response function of the units $g(x)$.

thus the output lies in the interval [0,1]. The function g_j is sketched in figure 1.

The response function of the output and hidden units needs to be differentiable if gradient descent is to be used. In fact, the derivative has the form

$$\frac{\partial g_j}{\partial \phi_j} = 2f'_j \tag{1.6}$$

$$= 2f_j(1-f_j). \tag{1.7}$$

The derivative reaches its maximum value for $f_j = 0.5$ and approaches its minimum value as f_j approaches zero or one. Thus, since the amount of change in a weight is proportional to f'_j, weights will be changed by a greater amount when they connect to units which have output states close to 0. Another property of g_j is that it can never reach the extreme values of 1 or −1 unless ϕ_j becomes infinite. Thus for the purposes of comparing the actual output of the network with the target values, the criterion used was that if the output was within a certain tolerance (*tol*) of zero or one, then the unit would be considered as being in the state zero or one respectively.

The cost function to be minimized (error E) is given by

$$E = \sum_p E_p = \sum_p \sum_j \frac{1}{2}(o_{jp} - t_{jp})^2 \tag{1.8}$$

with o_{jp} and t_{jp} representing the actual output at a unit in the output layer and the target output for each picture respectively. Using gradient descent the weights are updated after each set of pictures presented to the network according to

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{1.9}$$

which, using the chain rule, becomes [8,5]

$$\Delta w_{ij} = \sum_p \eta \delta_{ip} g_{jp}, \tag{1.10}$$

with δ_{ip} for unit i being a term consisting of the error at the output propagated back along all the weights leading from i. It is given by

$$\delta_{ip}^{(out)} = 2f'_{ip}(o_{ip} - t_{ip}) \tag{1.11}$$

for output units, and

$$\delta_{ip}^{(hid)} = 2f'_{ip} \sum_k g_{kp} \delta_{kp} \tag{1.12}$$

for the hidden units, where k labels units which unit i directly connects to in the forward direction. η is a parameter which determines the speed of the descent for a particular gradient, and is referred to here as the step-size. A slight modification to equation (1.10) provides a more controlled descent:

$$\Delta w_{ij}(n+1) = \sum_p \eta \delta_{ip} g_{jp} + \alpha \Delta w_{ij}(n) \tag{1.13}$$

where n indicates the number of the learning cycle, and α is known as the "momentum" parameter.

2. Implementation

The algorithm was implemented in occam2, in real 32 bit arithmetic on transputer arrays. The experiments for section 3 were performed on T414 transputers, and the later experiments in section 4 were performed on T800 arrays using a backpropagation simulation program developed in Edinburgh physics department (see for example [6,7]). The transputer arrays make up the Edinburgh Concurrent Supercomputer Project, on which various other neural network simulations have been performed using such MIMD (and also SIMD on the ICL DAP) parallelization [1,2].

3. Experiments with the rounding problem

3.1 Description of the task

The learning algorithm was applied to the following task. The network is required, when trained, to be able to round-off a set of numbers applied to its input to zeroes and ones at the output. There is a corresponding output unit for each input unit. The numbers applied to the input lie in the interval [0,1], and outside the range $(0.5-r, 0.5+r)$, where r is the parameter defining the difficulty of the task. That is, the problem domain to be learnt consists of the following mappings for a difficulty r:

Input	Output
$[0.5+r, 1.0]$	1.0
$[0.0, 0.5-r]$	0.0

Thus the network can be trained to perform tasks which require differing levels of discernment. The easiest task takes the form of a one-to-one mapping of binary patterns input to output ($r = 0.5$), and the greater difficulties are found when numbers either side of 0.5 are very similar and yet have to be mapped to different extremes. So the closer r is to zero, the harder it should be for the network to adjust its weights to achieve the required function.

Since the nature of this problem is such that each element forming an input picture is totally independent of the other elements (i.e. this is an order 1 problem, in the terminology of Minsky and Papert [4]), the elements in the output picture should correspondingly be independent. The only dependence between input and output is between elements corresponding to the same positions in input and output pictures. With this restriction it is clear that the network should tend to alter its weights such that it forms large weights for non-intersecting routes from the input elements through the hidden layer to the corresponding output elements.

The experiment was performed on a three layer network: the input layer, which contained up to seven units; the output layer, which had the same number of units as the input layer and the hidden layer, which could contain up to 25 units. It is of course essential that the hidden layer has at least the same number of units as input/output in order for a solution to exist at all. If there are more hidden units in the hidden layer than are *required* to find a solution, then there are expected to be a larger number of possible solutions, and one of these will have to be chosen by the system. The choice can depend only on the initial random weights. Thus the system can descend into different global minima of the error surface, by starting off at different points on the surface. The spectrum of global minima includes solutions where routes between input/output pair involve varying numbers of hidden units, and also the cases where some hidden units are not used at all.

It is necessary to include all permutations of the elements of possible input pictures in the training set. In training, the network should be taught to round off all numbers (N) in the range

$$(0.5 + r) \leq N \leq 1.0 \quad \text{and} \quad 0.0 \leq N \leq (0.5 - r)\,. \tag{3.1}$$

In order to ensure this, writing $R_+ = (0.5 + r)$, and $R_- = (0.5 - r)$, the pictures at the input units are taken to be all the permutations of 0, 1, R_+ and R_-, with R's only present at one of the inputs per picture, an example is given in table 1.

It can be seen that for c input units the number of pictures required will be

$$N(c) = c2^c \tag{3.2}$$

thus the number of pictures required in the training set scales worse than exponentially with the number of input units. This is the main problem with teaching an analogue neural network to perform a function which requires it to examine each input element individually, rather than absorb an overall

unit 1	unit 2	unit 3
R_+	1.0	1.0
R_-	1.0	1.0
R_+	0.0	1.0
R_-	0.0	1.0
R_+	1.0	0.0
R_-	1.0	0.0
R_+	0.0	0.0
R_-	0.0	0.0

Table 1: Part of the training set for a three input unit network. The rest of the set is obtained by permutations of the columns.

'taste' of the input. The latter is clearly more in the spirit of Parallel Distributed Processing [9]. However, it is beneficial to use a task which can be defined as easily as this one. A similar (in fact worse) scaling of the training set with network size is encountered in the parity problem [8].

3.2 Performance of the basic algorithm

The success of the algorithm in finding a solution can be demonstrated by a plot of the progress of the total error at the output units as a function of the training cycle. This is shown for systems 2–2–2 (2 input, 2 hidden, 2 output units), 3–3–3 and 4–4–4 in figure 2. On each graph is plotted the progress of total error with training cycle for each system for a certain difficulty r (0.5, 0.01, and 0.0001), with $\eta = 0.1$ and $\alpha = 0.6$. When the error remains essentially steady it is assumed that the algorithm is unable to converge to a global minimum, and has settled into a local minimum. It can be seen how, for a difficulty of 0.5, all the systems manage to locate a global minimum within a reasonable period. The descent is marked in all the systems by a relatively steep descent for the first 10 to 100 epochs, followed by a region of low gradient until the end. For the second difficulty, 0.01, only two systems manage to locate a global minimum. The descent is marked again by a steep fall in error during the first 10 to 100 epochs, but this time the almost level descent which follows is terminated by another relatively steep drop at 100 to 1000 epochs. The 4–4–4 curve is characterized by a rapid drop at the end, indicating the location of a sudden steeper descent, leading ultimately to a solution. It can be seen that the 3–3–3 system, however, does not locate a similar feature, and is destined to remain stuck on a plateau-like surface. With the third difficulty (0.0001) none of the systems manages to find a solution. The relatively steep initial descents are terminated at 10 to 100 epochs by a very flat portion, which shows no sign of ending.

Graphs were also plotted showing the terrain in the direction of motion of the system through weight space. The direction of motion is defined by the direction of the vector $\Delta\mathbf{w}(n)$. The terrain is mapped by calculating the error at various points forward and back from the present position on

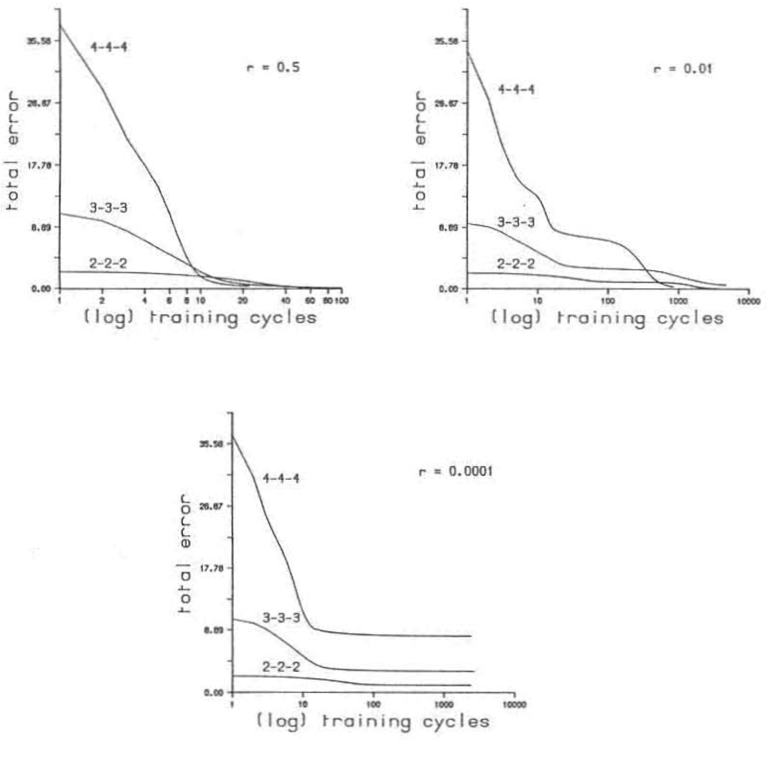


Figure 2: The progress of error with training cycle. Here we show three sizes of system at three task difficulties (r).

the error surface, in the direction of the next step to be taken. On some of these graphs an asterisk indicates the present position of the system, and a vertical line indicates the actual step to be taken by the system. This way of studying the error surface was suggested in [5], in which it was also shown how an initially large value of the momentum parameter can be dangerous, due to the large weight changes this causes at early stages of descent, when the error surface is steep.

The top graph in figure 3 shows a 2–2–2 system learning a mapping of difficulty 0.0001. The initial descent, shown in figure 2, is reasonably quick, but before long the error becomes quite stationary. The graph maps the terrain as the system clearly begins to iterate to the bottom of a local minimum. It is possible to make the system climb out of this by giving it a large step size. However there is no reason why the direction it takes should be one which brings the system to the brow of a hill, indeed the normal scenario is for the system to climb up to another local minimum, or a plateau, and stay there (bottom graph in figure 3).

The descent for the simplest system (2–N–2 with $r = 0.5$) is shown for the first six training cycles in figure 4. The terrain for the system with two hidden units is compared with that for the system with 25 hidden units. It can be seen how much steeper the descent is when there are a large number of hidden units. A direct comparison of the terrain at the start of each training session is shown in figure 5, for three difficulties. It can be seen that the line is steeper for networks with more hidden units. The reason for this can be understood by considering the first epoch of each system. If we assume that the weights (numbering 12 and 127 respectively) are of equal importance in the early stages of learning, and each weight is made to change in the learning algorithm such as to reduce the error, and if this change is small for each weight, then since

$$\frac{\partial E}{\partial w_{ij}} \propto \delta w_{ij}, \tag{3.3}$$

we have

$$\frac{dE}{dw} \propto \sqrt{\sum_{\{ij\}} (\delta w_{ij})^2} \tag{3.4}$$

and, if there are N weights in the system,

$$\frac{dE}{dw} \propto \sqrt{N}. \tag{3.5}$$

In figure 4 the gradient of the lines for the first epoch are 0.64 and 2.16 for the smaller and larger systems respectively. The ratio is 3.36, and, from the simple derivation leading to equation (3.5), we expect the ratio to be $\sqrt{127/12} = 3.25$.

Figure 6 shows the same situation for a larger network. However, here the surface is so much more mountainous anyway that the beneficial effect of

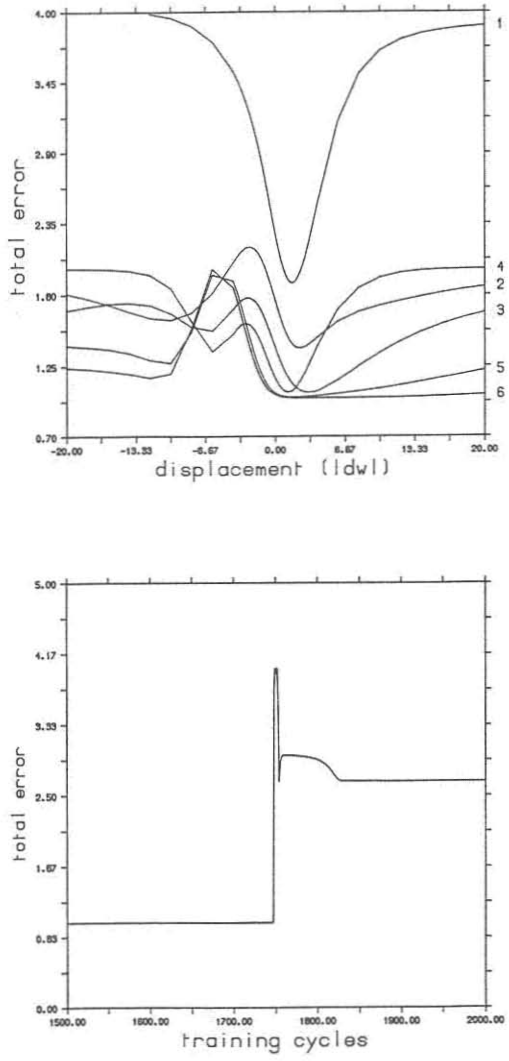


Figure 3: A descent into a local minimum, and an unsuccessful attempt to escape from one by momentarily increasing the step size. In the top graph each curve represents the terrain about the system for a new cycle, with the order of the curves indicated. The direction in weight-space plotted is that of the next weight-change (i.e. steepest gradient).

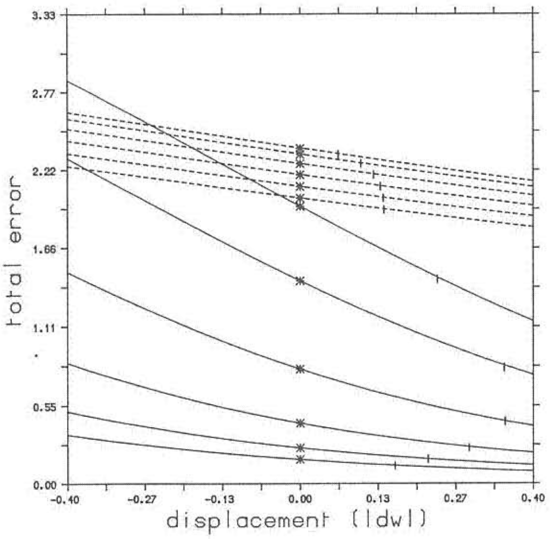


Figure 4: Comparison of the terrain for the first six epochs of descent in a 2–2–2 and a 2–25–2 system. The lines for the 2–2–2 system are dashed and those for the 2–25–2 system are solid.

the extra hidden units is best seen by considering the cliff-like terrain of the 7–25–7 network, as opposed to the valley-like terrain of the 7–7–7 network. The cliff-like descent is much quicker and more penetrating.

The addition of hidden units clearly has a consistently beneficial effect on the speed, and stability, of the descent.

In figure 7 we observe the effect of extra hidden units on the learning, at difficulty 0.5, for different network sizes. The graphs show the number of epochs to solution for each system size, averaged over 50–100 runs with different random starts. The error bars give some idea of the variation in learning time depending on a particular starting point on the error surface. It can be seen that for each of the networks shown, there is a definite trend for a quicker descent as the number of extra hidden units increases. Also there is possibly a trend for the addition of one or two hidden units producing a more dramatic effect as the network size increases. In all cases the addition of more hidden units has less effect as the total number of units in the hidden layer increases.

The influence of the momentum parameter is sometimes very important for finding a solution. For example, on a run with a 7–25–7 system with zero momentum a solution was reached after only 6 training cycles ($r = 0.5$), whereas when the system was trained with a momentum of 0.6 a solution was not found until more than 50 epochs. The descent during the initial few

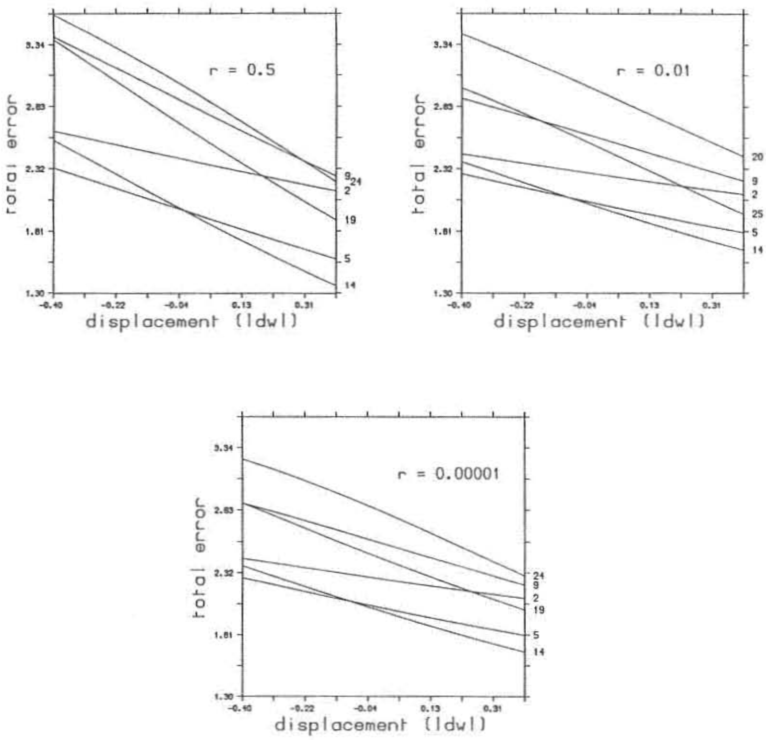
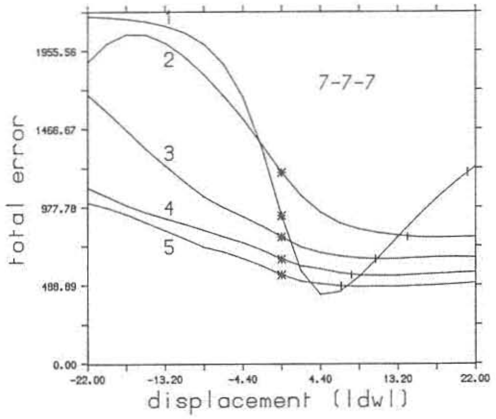


Figure 5: Terrain at the start of a training session for the 2–N–2 systems at three task difficulties. The value of N (number of hidden units) is indicated on the graphs, to the far right of each line.



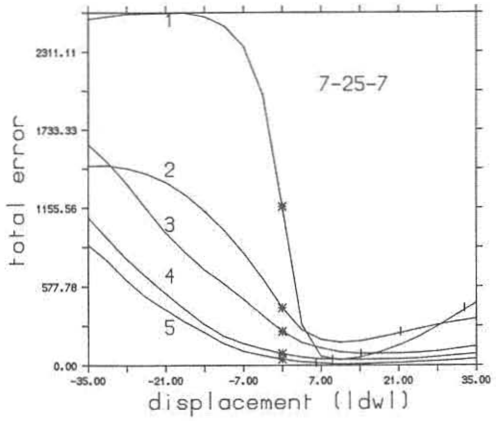


Figure 6: The terrain during the first five epochs at $r = 0.5$. Shown here are the 7–7–7 and 7–25–7 systems.

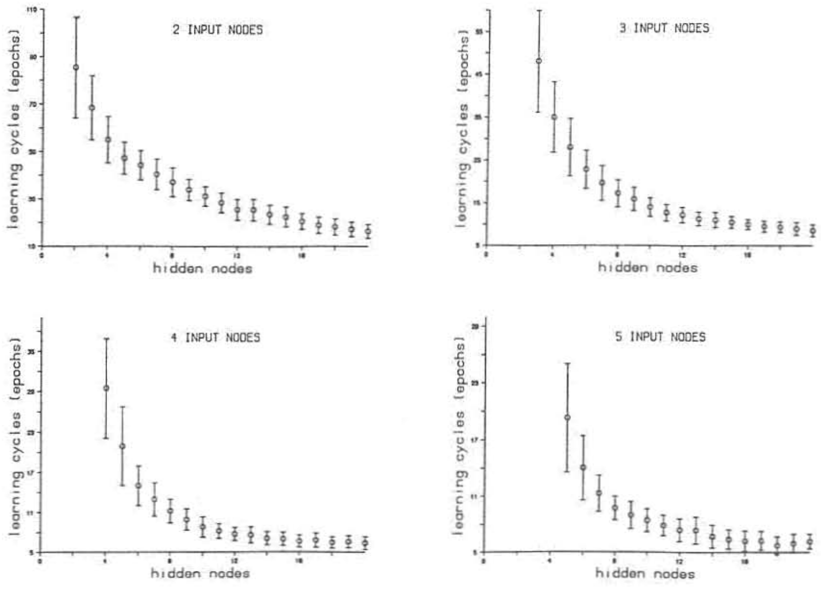


Figure 7: The effect on learning time as hidden units are increased. Error bars indicate the spread of learning times for different random starts.

epochs was quite similar for the two cases, however, at an error of about 3.1 the second system landed on a plateau. The following 40 or so epochs were taken up by slow progress along this plateau, until the cliff-like edge was found. This descent was interesting, and so the surrounding terrain for the relevant epochs was mapped out (figure 8). The graph shows how flat the plateau is and how steep the plunge at the end is. Comparison of this graph with the 7–25–7 descent in figure 6 shows how small a crack was actually found by the system. It seems that the presence of the momentum caused the system, by chance, to jump onto the plateau rather than continue with a reasonably comfortable descent. It is this kind of unpredictable behaviour in solving tasks such as this one which makes the gradient descent learning algorithm unattractive. The 'deformation' suggested below is designed to avoid this, and also to improve greatly on the level of difficulty which can be solved by the system.

The learning algorithm failed to find a solution for the larger networks, and smaller r, and just converged to local minima. Sometimes the system was helped by interactively altering step size and momentum at various stages in the learning. However, this was not considered to be a very satisfactory way of pursuing a better learning procedure, since the error surface could not be predicted for an arbitrary network at any particular point in the learning. Below we introduce two ways of improving the performance of the algorithm.

3.3 The deformation technique

The idea of deformation is to start the system off by training it to learn a relatively straightforward mapping task, and gradually to increase the difficulty of the mapping, in such a way that the task is eventually *deformed* into the task of the desired difficulty. This process can be viewed as a "topological" deformation[1] of a problem which can be represented as a simple shape in some space, into a more difficult problem whose extra difficulty is represented by the same topological surface, forming a more complex shape in the same space. Clearly, the class of problems which can be described in such a way will have to be defined if this method is to have any use, but for easily-defined tasks like the one studied in this paper, the deformation parameter is easy to identify, and such a method would appear to be a reasonable one to adopt. We give an example of another way of applying deformation in section 4.

The entire problem is completely defined by the error surface in multidimensional weight-space. The harder the problem one requires the network to solve, the more treacherous will be the terrain of the error surface, and the harder it will be for the system successfully to descend into one of the global minima of the surface. Thus one can picture the deformation procedure as moulding the error surface about the point occupied by the system, as the system descends towards the point of the final global minimum. In

[1]Here the use of the word topological is not meant to imply any rigid mathematical basis for this technique. Its use is mainly as a guide to understanding how the error surface is gradually altered.

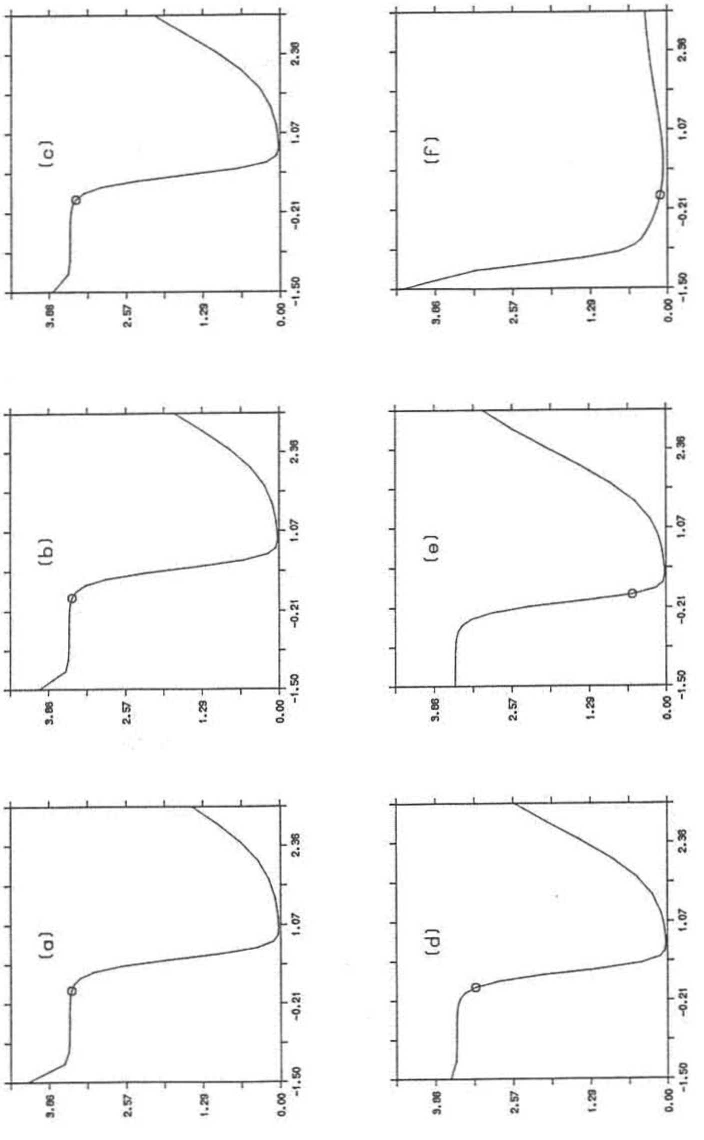


Figure 8: Location of a cliff-like edge in the gradient descent of the error surface of a 7–25–7 system with $\alpha = 0.6$. The figures show the progress of the network (the circle) as it comes to the end of a long slow plateau. No such problem was encountered when α was set to zero.

this way the system is able to avoid a lot of the treacherous terrain it would have to descend were it started off at a random position on the final error surface. This technique does not guarantee descent to a global minimum; the difficulty used at the beginning, and the parameters used to vary the deformation, need to be such that the surface can be gently deformed, with the task required to be learnt to vary smoothly at each deformation. This method can be compared with the simulated annealing technique [3] in which the system is eased into a global minimum of the surface defined by the cost function by reducing the noise of the system down to the value it has in the actual problem. The idea here is that by starting the surface descent at a high noise value (or high temperature), the system will tend to locate the global minimum from the beginning, and as temperature is reduced will remain within the basin of attraction of the global minimum. The difference between the two is that the annealing prevents the system from becoming trapped in local minima, while the deformation removes the need for the system to descend a hazardous surface, by moulding the surface around the system.

The problem at hand is clearly an ideal candidate for the deformation method. The deformation parameter is r, which is to be decreased in stages from 0.5 to a final value r_0.

3.3.1 Deforming the error surface for the rounding problem

The difficulty can be varied continuously in the rounding problem, so one needs to determine the change in r required as a function of r. For a problem with a discrete set of difficulty levels it may be a simpler matter to determine such a schedule. Initially r was changed by a constant factor (0.99) each time. It was found that the factor was required to be closer to unity as the r decreased in order that the system remained near enough to the bottom of a "ravine-like" structure in the error surface so that it did not break out of it into some local minimum. Thus it is necessary to find some way of getting the r-change to cause an alteration in the error surface which is sufficiently small that the new error is not significantly different from that attained after completion of learning for the old r. The expression derived below informs us of the change in position of the system on the error surface (E) after it has been deformed due to a change in r.
The error defined in equation (1.8) is rewritten as

$$E = \sum_p \sum_i E_{ip} \tag{3.6}$$

$$\text{where} \quad E_{ip} := \frac{1}{2}\left(t_{ip} - o_{ip}\right)^2 . \tag{3.7}$$

After a difficulty has been learnt the system is able to round numbers outside that r to zero or one respectively (within a tolerance *tol*). Thus for a single output unit and a single picture the maximum error at the end of a deformation cycle is given by

$$E_{max} = \frac{1}{2}(tol)^2. \tag{3.8}$$

It is wished to control the change in r such that the error at this output unit increases by the same (tolerable) amount each time. The assumption is that the error at the other units, and for other pictures, will behave similarly, or at least no worse than that at the output unit with the maximum error.

In deriving the expression for $\frac{\partial E_{ip}}{\partial r}$ for a three-layer network we use the following notation:

input units are labelled by the subscript n

hidden units are labelled by the subscript j

output units are labelled by the subscript i.

Thus

$$E_{ip} = E_i(o_i) \tag{3.9}$$

dropping the subscript p, and so

$$\delta E_i = \frac{\partial E_i}{\partial o_i}\delta o_i \tag{3.10}$$

where

$$\delta o_i = \sum_j \frac{\partial o_i}{\partial g_j}\delta g_j \tag{3.11}$$

for fixed values of the weights. Similarly

$$\delta g_j = \sum_n \frac{\partial g_j}{\partial g_n}\delta g_n. \tag{3.12}$$

Using equations (1.2) and (1.5), eqns. (3.11) and (3.12) become

$$\begin{aligned} \delta o_i &= \sum_j \{w_{ij} o_i (1 - o_i)\, \delta g_j\} && (3.13) \\ \delta g_j &= \sum_n \left\{ w_{jn} \frac{(g_j + 1)(1 - g_j)}{2} \delta g_n \right\}. && (3.14) \end{aligned}$$

From the definition of E_i we have

$$\begin{aligned} \delta E_i &= -(t_i - o_i)\, \delta o_i && (3.15) \\ &= -\sqrt{2E_i}\; \delta o_i. && (3.16) \end{aligned}$$

Due to the pictures that are presented to the system, δg_n is only non-zero for one of the input units per input picture, and always has the values:

$$\begin{aligned} \delta g_n &= +2r \quad \text{for} \quad t_{i=n} = 1 && (3.17) \\ \delta g_n &= -2r \quad \text{for} \quad t_{i=n} = 0. && (3.18) \end{aligned}$$

picture	δ^{out}	δ^{hid}
R_+	0.00988	0.002445
R_-	-0.00988	-0.002446

Table 2: Values of δ for $r = 0.011$ for the system in figure 1.

Now, substituting eqns. (13) and (14) in eqn. (16) we find

$$\begin{aligned} \delta E_i &= -\sqrt{2E_i}\, o_i(1-o_i)\sum_j \left\{ w_{ij}\frac{(1+g_j)(1-g_j)}{2}\sum_n w_{jn}\delta g_n \right\} && (3.19)\\ &= -2E_i\left(1-\sqrt{2E_i}\right)\sum_j \{(1+g_j)(1-g_j)w_{ij}w_{jn}\delta_{ni}\}\,\delta r, && (3.20)\end{aligned}$$

where δ_{ni} is here the Kronecker delta. Hence the change in error with r is given by:

$$\frac{\partial E_i}{\partial r} = -2E_i\left(1-\sqrt{2E_i}\right)\sum_j \left\{\left(1-g_j^2\right) w_{ij}w_{jn}\delta_{ni}\right\}. \qquad (3.21)$$

In other problems the variation might be more obvious. (For example, a basic picture of an article of clothing might be shown, followed by a set of progressively more unusual or highly decorated versions of such an article. In learning to recognize a whole range of clothing, the basic object is understood first, in its essence, rather than presenting the whole set all at once and expecting the net to organize sensibly from the start.) In the simulations below, equation (3.21) was used in determining the amount by which r should be changed after each stage in the deformation had been successfully learnt. The maximum tolerable error change (δE) was taken to be 0.0005 for all simulations (with $tol = 0.1$), however this value is not critical.[2]

3.4 Speeding up the descent (using momentum safely)

The deformation process was very successful in allowing networks to solve tasks of much greater difficulty ($r_0 = 0.0001$), however it became clear that the updating procedure for the weights became more inefficient as r was decreased. This can be demonstrated with a simple example where the network has one unit in each of its three layers (see figure 9). The system has just learned at the deformation stage of 0.005, and is about to start error propagation at the next r of 0.004881. The numbers which are presented are: 0.504881 (R_+) and 0.495119 (R_-), with the state of the threshold unit always at 1.0. The total error at the output unit at the end of the last r is 0.01. Tables 2 and 3 show the δ's and gradients at this point in the training of the system.

[2]It is important for δE not to exceed an upper limit (so that the system stays in the ravine), while an optimum value is determined by the minimum number of cycles required to learn at the new r.

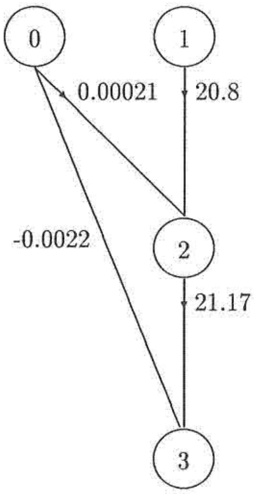


Figure 9: Example of the weight situation in a 1–1–1 system at a particular point in the learning. The numbers on the lines indicate weight values and those in the circles label units.

weight	gradient
w_{20}	-0.000001
w_{21}	0.0000477
w_{30}	+1.0E-08
w_{32}	0.00200

Table 3: Gradients for the weights in the system in figure 1 (r = 0.011).

It can be seen how inefficiently the large weights are updated. The reason for this small update, despite the comparatively large δ's, is contained in the expression for the gradients

$$\frac{\partial E}{\partial w_{ij}} = -\sum_p \delta_{ip} g_{jp}. \tag{3.22}$$

The values of the δ's remain similar as r is decreased, since the deformation ensures the system remains close to the tolerance error, while the values g_{ip} *decrease* with decrease in r. Thus the time (in learning cycles) taken to learn each new r unavoidably increases as the system learns to round numbers closer to 0.5. This is a feature of the kind of task studied, and shows the inefficiency of this method of back propagation for such a task, as the input pictures (which are to be distinguished by different mappings) become more alike. Steps were taken to try to speed up the learning, and it was found that the acceleration provided by the momentum parameter was very effective — *provided the acceleration was suitably controlled.* The method of controlling the speed of descent will be described below, after an example of a hazard which exists during gradient descent in a valley.

Figure 10 shows how the system climbs up a valley using gradient descent by bouncing from one wall to the other. The explanation for this effect, which ultimately leads to the system hanging on a flat region outside the valley, is that the step size at the first (lowest) point is just too big at that point on the valley wall to produce a weight change which will send the system down the valley. Thus the system finishes at a point higher up on the opposite valley wall. It might be expected that with its next step the system would have rectified this, there being less chance of the weight change being so large that the same occurrence is repeated. However, this is hardly ever the case, due to the effect of deformation on the *shape* of the valley. This will be illustrated below.

The above "valley ascent" was found to be most undesirable (the outcome is that the system arrives at the top of the valley and gets stuck on a level plane) and decided the fate of all system sizes below a certain value of r. To combat this it was decided to reduce η at the point this behaviour was detected. Thus η now becomes dependent on the direction in weight space. The onset of the valley ascent is marked by two weight changes in opposite directions, the second of which has a greater magnitude than the first. When this is detected η for that weight is reduced.

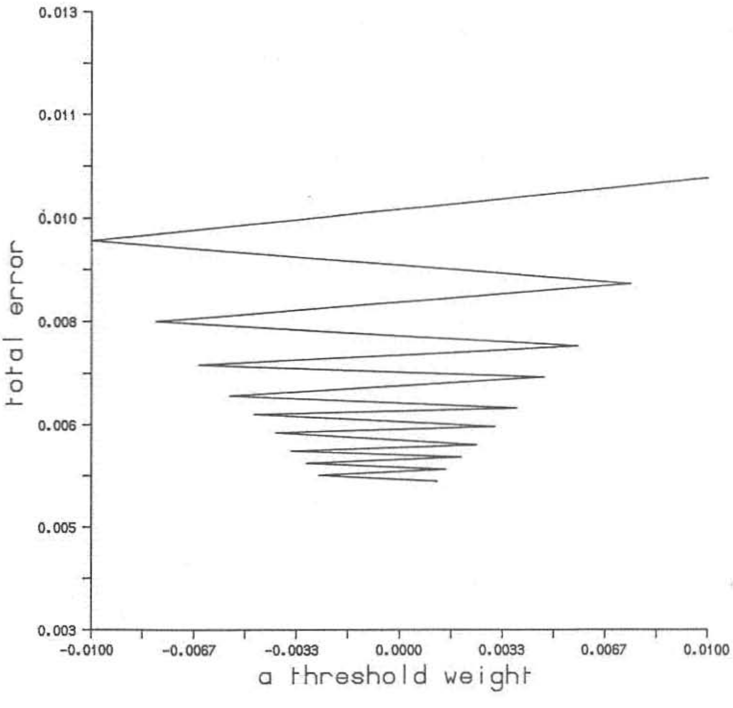


Figure 10: Ascent from an error minimum using the gradient descent algorithm. The scenario begins at the lowest point shown, with each following calculation causing an increase in the total error.

Momentum was found to be indispensable as a way of speeding up descent of slightly sloping regions (which characterize error surface in the directions of the heavy weights), and also for descent down valley walls when step size is small. However, it is important to ensure that momentum is 'switched off' whenever the descent reaches a stage at which it crosses the valley bottom (adding the previous weight change after this would result in ascent of the opposite wall). This is recognized by the gradient having the opposite sign on opposite valley walls.

By carefully controlling the speed of descent in this way, it was possible to solve tasks down to very small values of r (10^{-7}–10^{-8}), for all the systems studied.

3.5 The shape of the error surface at various stages of deformation

In order to investigate the shape of the error surface as the system made its way down to a global minimum, for a particular stage in the deformation, maps were plotted in different weight directions (i.e. the learning was halted and a particular weight was varied to observe the variation in error produced). The various cross-sections (figures 11, 12, and 13) reveal three different landscapes, characterizing three different types of weight. These are the threshold weights, the heavy weights (forming the non-intersecting paths), and the remaining weights. Also shown is the changing shape of the error surface as deformation proceeds. It can be seen that at very low values of r the environment in the direction of a threshold weight becomes more and more crevasse-like. Herein lies the reason for the valley ascent described in the previous section; as r is decreased the valley into which the system descended at the beginning of the deformation (the initial problem) is transformed into one with progressively steeper walls. If at any point the system should now climb up a little bit, future gradient calculations would produce enormous weight changes, thus sending the system bouncing from wall to wall up the valley, eventually to leave it stranded on a level plane.

The form of the error surface in the other directions in weight space, as the deformation proceeds, is also interesting. The heavy weight directions reveal a gently sloping error surface, which becomes flatter as the deformation continues, as would be expected. The error surface in the directions of the remaining weights is somewhat surprising. It hardly changes from its original shape of a gentle flat-bottomed valley, even when r gets as low as 10^{-6}. Evidently the learning procedure is not as sensitive to the value of these weights as it is to the threshold weights.

It becomes apparent from the maps how effective, and desirable, the deformation technique is. A system starting off at $r = 10^{-5}$, for example, never finds the crevasse-like structure into which it has to descend, and even if it did would have extreme difficulty in remaining in it. The deformation process makes things easier for the algorithm at the start, and then 'eases' the system into the difficult terrain characterizing the original problem.

The maps also show that it is the threshold weights which are the ones

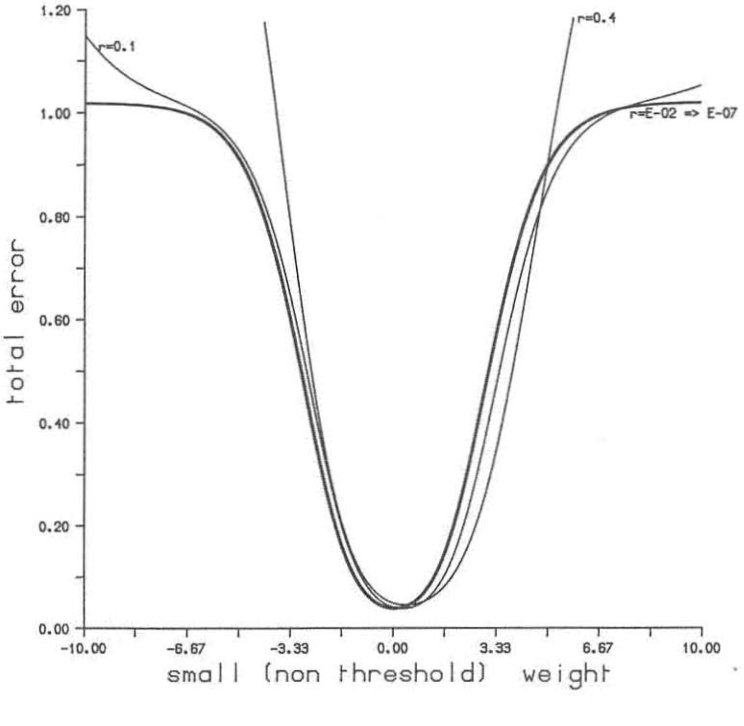


Figure 11: Error surface in a small (non threshold) weight direction, at different stages of deformation (r). The learning is halted at a particular r, the value of the weight is varied and the new error at each value calculated.

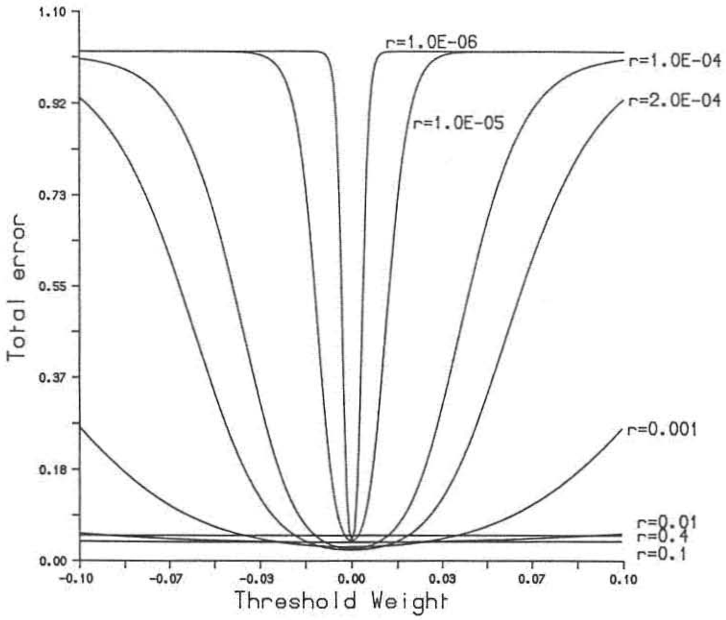


Figure 12: Error surface in a threshold weight direction, at different stages of deformation.

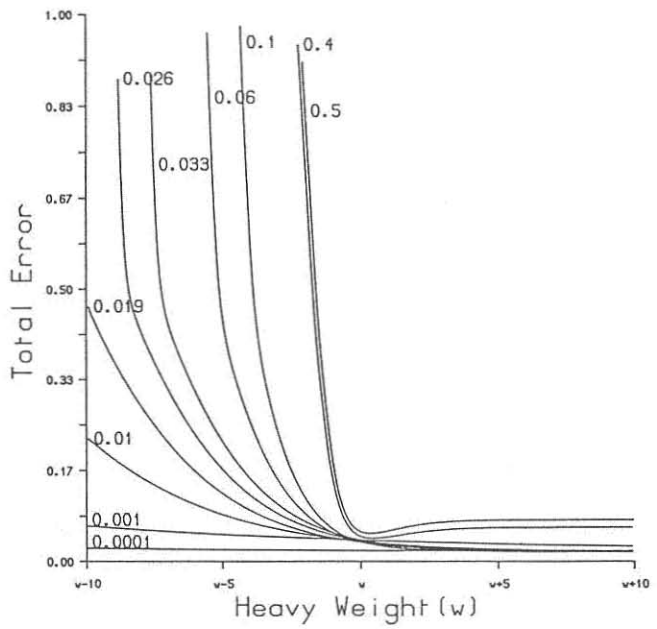


Figure 13: Error surface in a heavy weight direction, at different stages of deformation.

most crucial for the stability of the system, and that even a very small deviation from zero causes several outputs to be totally wrong — that is, close to 1 when the target is 0 (each of these 'flippings' is characterized by an extra error of 0.5). It is the form of the response function which produces the level planes after these flippings.

3.6 Hidden unit redundancy

With the addition of more hidden units, it becomes difficult to analyse the patterns of heavily weighted routes from input to output, since this typically includes more than one path for each input/output pair. Thus it is useful to represent the system graphically, to provide an indication of the routes taken. After each learning cycle, when the weights were updated, the intensities of lines on the graphics screen, representing weights, were updated. The intensities were normalized to the weight with the greatest (absolute) value. On the screen negative weights were blue, and positive weights green. This provided us with 125 different intensities in each colour, a reasonable indication as to the relative strengths of the weights. Weights of negligible size compared with the larger weights have negligible intensities. Analyses of various sizes of system showed that one frequently obtained hidden units with negligible weights to and from all output and input units. An example is shown in figure 14, a screen dump of a 5–15–5 network which has learned down to a range of 10^{-4}. Similar patterns are observed in other networks with large numbers of hidden units. It appeared that such occurrences were the results of competition between two or more input/output routes of similar strength resulting in a draw, with the weights concerned subsequently becoming negligible compared with weights in other routes, and the routes themselves thenceforth abandoned.

4. Using deformation to learn noisy patterns

We applied the deformation technique introduced in the last section to another problem domain. The purpose is to demonstrate another type of deformation parameter — degree of noise in binary images — and to illustrate how the technique produces much faster and more controlled learning.

The network used has a 45–10–45 architecture. The input and output layers are to be viewed as 5 by 9 arrays of pixels. The input units themselves take on only the binary values 1 or 0. The training set consists of a set of noisy images of digits which are to be mapped to their corresponding clean images at the output. The difficulty of the problem is a function of the amount of noise present in the inputs, since the greater the noise the less the basic structure of the digit is seen. Thus one can imagine the error surface becoming very hazardous at various points, especially when the training set contains digits already very highly correlated without noise.

The training schedule is clear: teach the network first of all the clean images (i.e. N–H–N encoding), and then introduce noise at the input patterns,

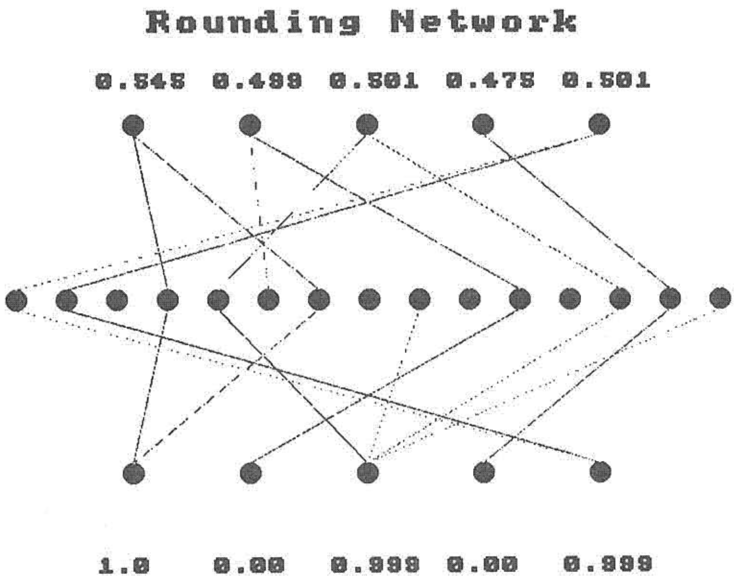


Figure 14: Screen dump from a graphics display, during the training of a 5–15–5 system. The numbers at the top and bottom of the network indicate the progress of the rounding. The presence of dominant weights and redundant units can clearly be seen.

% noise	final error	% patterns correct
5	20.018	80
10	20.021	80
15	20.027	80
20	20.032	80

Table 4: Performance of the basic algorithm.

until the desired noise value is obtained. That is, the final operation of the net is to be one in which for any digit corrupted by noise of value less than or equal to $n\%$, a clean image of a digit will be produced at the output. For relatively large noise values it may be the case that the noisy image of a particular digit is "closer" (in terms of a distance measure the net is using) to another digit. In this case the net should produce as output the second digit. The network can be viewed as a device (characterized by a particular noise tolerance n) which cleans up noisy images by producing at output the digit which is closest, in terms of general structure, to the input image.

It is clear that the error surface for such a functionality is necessarily very highly structured and will contain many crevices and steep descents.

First we observe the performance of the basic algorithm on the 5%, 10%, 15%, and 20% noise domains. Each training set consists of ten examples of each digit, that is 100 patterns in all. The learning parameters used here and in all subsequent runs are $\alpha = 0.9$, $\eta = 0.1$, $tol = 0.15$.

The network was not able to achieve 100% success in any of the noise categories. (a run was terminated after 10,000 epochs, when the rate of change of error was slower than one part in a thousand per epoch). Table 4 shows the performance of the network in terms of the percentage patterns correct. We show in figure 15 typical ways in which the network got stuck. The figures show the input, hidden and output unit states for a particular pattern in the training set. In one case all the mappings were correct apart from all the 1's with one pixel wrong, and all the 2's with the same three pixels wrong. This type of error is similar to the "flipping" in the last section, and is due to the large gradients in the error surface. Clearly, certain patterns are very similar to each other, and the net is most likely to descend into a local minimum giving rise to mixture states. The minimum the net is required to reach probably becomes either narrower or further away (or both), the more noise that is present.

Next three deformation procedures were tried. The first involves the sequence clean $\rightarrow$ 10% $\rightarrow$ 20%, and the second used the sequence clean $\rightarrow$ 5% $\rightarrow$ 10% $\rightarrow$ 15% $\rightarrow$ 20%, and the third clean $\rightarrow$ 20%. It was not attempted to find an optimal deformation schedule for learning up to the 20% noise training set. These experiments were done to demonstrate the suitability of the deformation procedure for this type of problem. It is not even necessary to use such a hard problem; as was suggested above the idea

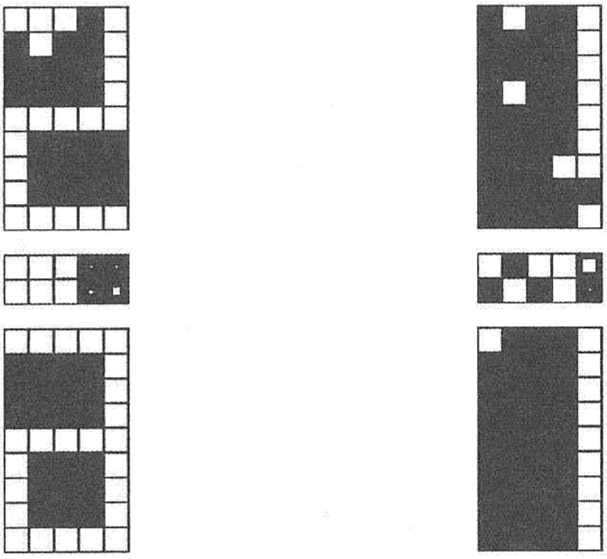


Figure 15: Examples of where the basic algorithm got stuck in the learning of noisy digits. The input units are at the top and the outputs should show the letters "2" and "1" respectively. Such errors were made even at the 5% noise level shown here.

% noise	final error	accumulated epochs	% patterns correct
0	0.467	1051	100
10	1.021	1430	100
20	0.608	2351	100

Table 5: Deformation schedule 1.

% noise	final error	accumulated epochs	% patterns correct
0	0.467	1051	100
5	1.122	1300	100
10	1.021	1562	100
15	0.302	3361	100
20	1.505	15,000	99

Table 6: Deformation schedule 2.

is more to build on current more general knowledge in a sensible way. We show below that deformation enables the network to find very good minima in a hazardous error surface. Deformation may help even when a global minimum may not exist (i.e. in the cases when there are conficting members present in the training set), by keeping track of the optimal minimum using previous knowledge. Tables 5, 6, and 7 show the performance of the net for each deformation schedule. Using the first schedule the network was able to learn successfully all the training sets. Typical mappings for the 20% noise network are shown in figure 16. Using the second or third schedules the net was not able to complete the learning, but the local minima in which it got stuck are much lower than for the basic net. Actually nearly all the patterns were correct. We show an example of an incorrect mapping in figure 17. The optimum deformation schedule lies somewhere between the second and third schedules tried above.

Looking at figure 16 again, it can be seen how the net performs the mapping of apparently quite different noisy images of the same digit, by responding to the features in the image which are most typical of the digit. This can be seen in the activations in the hidden layer for patterns in the same digit class. This representation in the hidden layer is then used to reproduce the digit at the output layer. Without this two-level processing capability networks would not be able to perform most interesting tasks involving extraction of the relevant information from the activations at the input.

5. Summary

We have used the rounding problem to investigate various aspects of the learning procedure of a feed-forward net. This problem domain was useful in that its difficulty could be continuously varied. The basic algorithm was

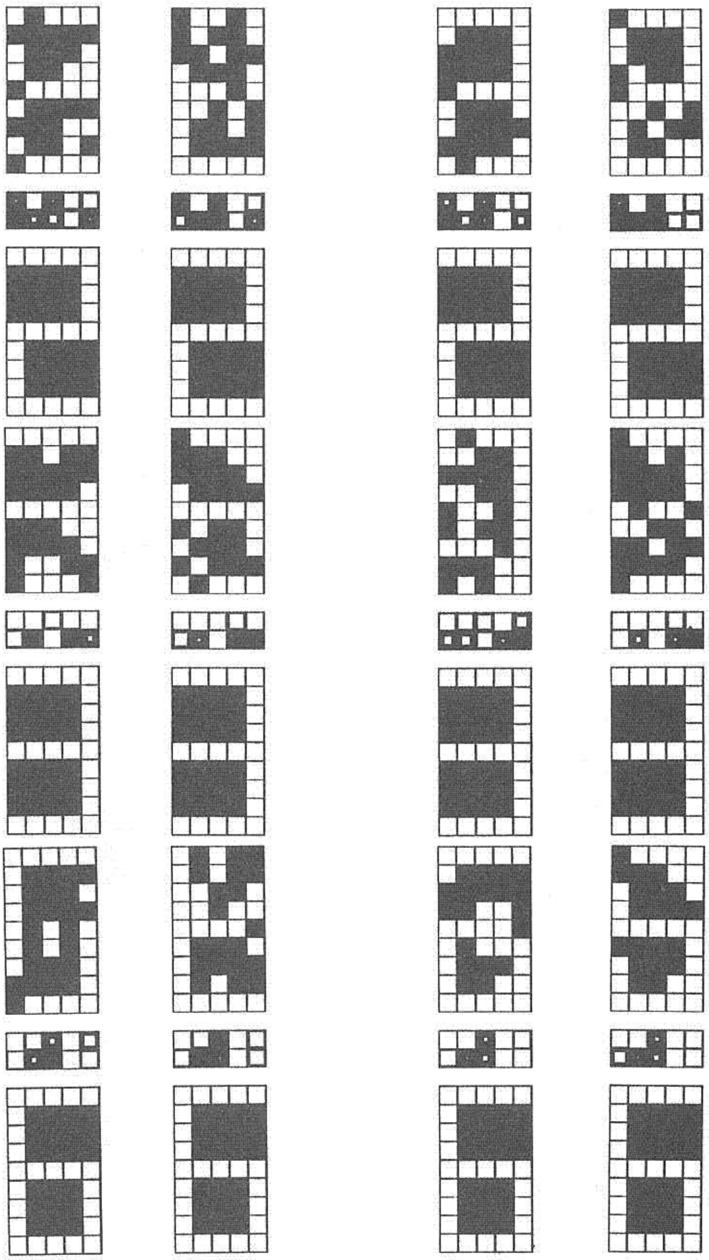


Figure 16: Examples of the successful hard mappings learnt by the network using the deformation technique. Shown here are four examples of mappings of noisy versions of the digits "2", "3", and "6". The noise here is 20%.

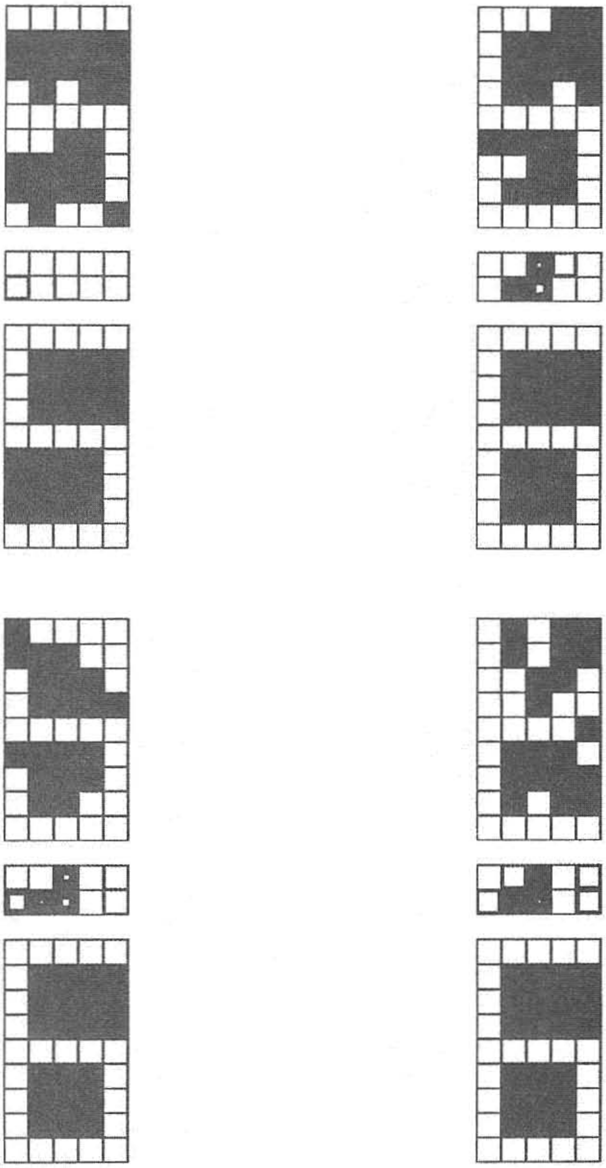


Figure 17: The incorrect mapping at which the net using the deformation technique got stuck. In one instance of mapping of noisy 6's the net maps to a "5". Note how this error shows up in the hidden layer. All other patterns were successfully mapped at this noise value of 20%.

% noise	final error	accumulated epochs	% patterns correct
0	0.344	1160	100
20	1.563	5000	99

Table 7: Deformation schedule 3.

found to experience much difficulty as the problem difficulty was increased. Observation of the characteristics of the error surface led to the development of the deformation procedure and a reliable way of varying the step-size during learning so that momentum could be used more effectively in accelerating the gradient descent. With these modifications to the training procedure the net was seen to complete learning to a satisfactory level for all the systems considered. The scaling of the learning procedure to larger networks was favourable (in terms of epochs taken, not training set size and actual compute time), and it was found that extra hidden units speeded up the learning. It was also noticed that sometimes hidden units were not used very much, pointing to a redundancy among units.

An example of another application of the deformation technique was given in the association of sets of noisy images of digits with their clean images. Use of the deformation technique was shown to improve vastly on the basic learning algorithm. This example also demonstrated the way in which the network uses the hidden units to represent the salient features of an image, so that it can produce the required mappings.

The lesson to be learned from the success of techniques like deformation, is that however attractive pure self-organization may sound, it is emminently more sensible and worthwhile to place at least a few basic constraints on a network during the learning, to retain a little control in the outcome of net learning.

Acknowledgements

The authors would like to thank David Wallace for suggesting the rounding experiment, and Elizabeth Gardner for many useful discussions.

References

[1] B.M. Forrest, D. Roweth, N. Stroud, D.J. Wallace, and G.V. Wilson, "Implementing Neural Network Models on Parallel Computers", *Computer Journal*, **30** (1987) 413–419.

[2] B.M. Forrest, D. Roweth, N. Stroud, D.J. Wallace, and G.V. Wilson, "Neural Network Models", *Proceedings of Vector and Parallel Processors in Computational Science III*, Liverpool (August 1987)

[3] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, **220** (1983) 671–680.

[4] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, 1969).

[5] D.C. Plaut, S.J. Nowlan, and G.E. Hinton, *Experiments on Learning by Back Propagation*, Carnegie Science Department, Carnegie-Mellon University, preprint no. CMU-CS-86-126 (June 1986).

[6] G.D. Richards and F. J. Śmieja, *ECS Newsletter* (March 1988).

[7] G.D. Richards, *ECS Note* ECSP–UG–7.

[8] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation", *Nature*, **323** (1986) 533.

[9] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, Parallel Distributed Processing, vols. 1 and 2 (MIT press, 1986).

[10] A. Wieland and R. Leighton, "Shaping Schedules as a Method fo Accelerated Learning", presented in the *First Annual Meeting of the INNS*, Boston (September 1988).