LRP 398/90 December 1990

# IMPLEMENTATION OF A MULTI-LAYER PERCEPTRON FOR A NON-LINEAR CONTROL PROBLEM

J.B. Lister, H. Schnurrenberger, Ph. Marmillod

# IMPLEMENTATION OF A MULTI-LAYER PERCEPTRON FOR A NON-LINEAR CONTROL PROBLEM

JONATHAN B. LISTER, HARRY SCHNURRENBERGER,
PHILIPPE MARMILLOD

*Abstract* - We present the practical application of a 1-hidden-layer multi-layer perception (MLP) to provide a non-linear continuous multi-variable mapping with 42 inputs and 13 outputs. The problem resolved is one of extracting information from experimental signals with a bandwidth of many kilohertz. We have an exact model of the inverse mapping of this problem, but we have no explicit form of the required forward mapping. This is the typical situation in data interpretation. The MLP was trained to provide this mapping by learning on 500 input-output examples. The success of the off-line solution to this problem using an MLP led us to examine the robustness of the MLP to different noise sources. We found that the MLP is more robust than an approximate linear mapping of the same problem.

12 bits of resolution in the weights are necessary to avoid a significant loss of precision. The practical implementation of large analog weight matrices using DAC-multipliers and a simple segmented sigmoid is also presented. A General Adaptive Recipe (GAR) for improving the performance of conventional back-propagation was developed. The GAR uses an adaptive step length and both the bias terms and the initial weight seeding are determined by the network size. The GAR was found to be robust and much faster than conventional back-propagation.

# I. INTRODUCTION

The tokamak is an experimental device for studying magnetically confined hot plasmas close to the conditions required for thermonuclear fusion reactor ignition. One problem confronting research in this field is the precise definition of the plasma shape at any given instant. This information is essential for the closed-loop feedback control of the shape using many high power amplifiers. This extraction of the relevant data for control purposes must also be carried out fairly fast, with a bandwidth of up to a few kilohertz. The result must be precise since it ultimately limits the precision of the feedback control loop.

Figure 1 illustrates a cross-section of the DIII-D tokamak [1]. The solid closed line represents the shape of a surface of constant magnetic flux, as do the dashed lines inside it. The full and open circles represent the positions of magnetic field probes and magnetic flux loops respectively, which provide the experimental signals to be interpreted. The various arrows define some simple geometric properties of the plasma. The mapping we require is from the experimental input signals to these geometric parameters.

We must therefore develop a useful representation of the mapping between a measurement data vector **M** and an extracted parameter vector **G**. We shall denote this mapping by $G$, such that $\mathbf{G} = G(\mathbf{M})$. This particular equilibrium problem is typical of many data interpretation problems, for which we possess a full physical description of the inverse mapping $G^{-1}$, in our case the Grad-Shafranov plasma equilibrium equation. Since we cannot derive a closed form of the mapping G from this physical model, we are required to construct an approximation for G, denoted by $\hat{G}$, which we hope will be valid over the entire range of interest of the vector **M**.

In the case of almost fixed circular plasmas, an approximate mapping $\hat{G}$ is obtained by linearising, and then inverting, the known physical mapping $G^{-1}$. As the shape of the plasma varies and the range of equilibrium parameters increases, the linearised mapping becomes less and less reliable, especially for extreme values of plasma parameters, which are of most interest to us.

So far, there have been three main approaches to approximating the non-linear mapping $G$, which we briefly discuss for comparison.

The first approach is to develop a description of $G$ using linear ad hoc trial functions, and has been successfully used on the DIII-D tokamak [1-3]. A database containing many different tokamak shapes provides examples of both measurement (**M**) and parameter (**G**) data. Trial functions are proposed, intuitively or by inspection, and the several free parameters in each of the trial functions are fitted by regression analysis.

The second approach is to reduce the data by Principal Component Analysis (PCA) and subsequently develop a quadratic expansion for $\hat{G}$ [4] giving:

$$\hat{G}(\mathbf{M}) = \mathbf{A} + \underline{\mathbf{B}} \cdot \mathbf{M}^* + \mathbf{M}^{*T} \cdot \underline{\underline{\mathbf{C}}} \cdot \underline{\mathbf{M}}^* \quad ; \quad \underline{\mathbf{M}}^* = \underline{\mathbf{PCA}} \cdot \underline{\mathbf{M}} \tag{1.1}$$

where $\underline{\mathbf{PCA}}$ is the principal component projection matrix and $\mathbf{A}$, $\underline{\mathbf{B}}$, and $\underline{\underline{\mathbf{C}}}$ together provide a non-linear (quadratic) mapping. We are restricted to a relatively small dimension of $\mathbf{M}^*$ in order that the large tensor $\underline{\underline{\mathbf{C}}}$ be realistically implementable in hardware. One significant advantage over the method of ad hoc trial functions is the generality of the expression (1.1), allowing its straightforward implementation.

The third method presupposes no functional form for $G$ except that it be piecewise locally linearisable over a significant region in the space of **G**. The linearised $\hat{G}$, now a matrix, must be varied as the space of **G** is explored by the plasma evolution. Each local matrix $\hat{G}$ can either be deduced by linearising an appropriate subset of an equilibrium database, or directly from the equilibrium. The former approach fits a hyperplane through a portion of the function $G$, minimising the mean square residuals of the set of examples of $\mathbf{G}$. The latter approach calculates the local tangential hyperplane. These piecewise linear methods have been developed for the control of the TCV and Alcator C-MOD tokamak plasmas [5, 6].

In this paper, we present the use of a multi-layer perceptron (MLP) for approximating the mapping G(**M**). The quality of the representation is excellent, and the physics aspects of the problem have been described elsewhere [7]. In Section II we define the Neural Network which we have used. During the initial phase of this work we suffered from the slowness of conventional back-propagation. We therefore discuss our own improvements to the speed and automated use of back-propagation embodied in a General Adaptive Recipe (GAR) described in Section III. In Section IV we present the results obtained with the converged MLP, reproducing the required mapping for both a small test-case problem and real DIII-D experimental data. The results were very encouraging and so we followed this study by posing some practical questions concerning the use of an MLP for the non-linear mapping of tokamak data. Section V presents the hardware implementation of a fast 3000 weight hybrid analog-digital MLP with 12 bit resolution for the matrix weights. The number of bits of resolution necessary is examined in Section VI. In Section VII we look at the noise immunity of the MLP solution compared with a simple linearised mapping, and in Section VIII we study the error induced by a seven segment piecewise linear sigmoid to be used in conjunction with the hybrid matrices. Section IX provides a brief summary of the work.

## II. NETWORK CHOSEN

Since the mapping $G$ will be continuous and smooth except at a small (one or two) number of boundaries, a candidate choice for implementing a non-linear map of the type required is the feedforward Multi-Layer Perceptron (MLP) [8]. The MLP can be configured with one (MLP-1) or more hidden layers to produce different classes of non-linear map. Figure 2 shows an MLP-1 as a guide to the nomenclature in this paper.

Many authors have discussed the classes of functions which MLP-1 and MLP-2 networks can reproduce [9-12]. It seems that there is not yet a simple rule for determining the necessary configuration other than "use the simpler MLP-1 if it is good enough". The basic structure of the two networks is usefully illustrated in Fig. 3, in which we have tried to reproduce a cylindrical pillar in a 2-D input space. The MLP-2 network $(N_1{:}N_2{:}N_3{:}N_4) = (2{:}4{:}1{:}1)$ produces a clear pillar, whereas the MLP-1 (2:8:1) has a fringe around it which cannot drop off faster than $(\text{Radius})^{-1}$ outside the pillar as the number of hidden-layer neurons becomes large. The MLP-1 cannot reproduce a mapping in which there are large discontinuities in spatial coherence of the output without an impractical number of neurons in the single hidden layer whereas the MLP-2 can reproduce such functions with fewer neurons. On the other hand, the MLP-2 seems to take much longer to converge for a given problem. This may be largely due to the fact that the structure of the MLP produces large correlations between the weights in successive layers. For an MLP-1:

$$\frac{\partial \text{output}}{\partial W^{12}} = f(W^{23}, W^{12})$$

$$\frac{\partial \text{output}}{\partial W^{23}} = f(W^{12})$$

$$\frac{\partial^2 \text{output}}{\partial W^{12}\, \partial W^{23}} = f(W^{12})$$

which is already correlated, whereas for an MLP-2

$$\frac{\partial \text{output}}{\partial W^{12}} = f(W^{12}, W^{23}, W^{34})$$

$$\frac{\partial \text{output}}{\partial W^{23}} = f(W^{12}, W^{23}, W^{34})$$

$$\frac{\partial^2 \text{output}}{\partial W^{12}\, \partial W^{23}} = f(W^{12}, W^{23})$$

providing even more correlation between the successive matrices than the simpler MLP-1.

The MLP-1 was configured to be fully connected between adjacent layers, with linear outputs, illustrated in Fig. 2. Both input and output data were normalised to the range [-1, 1]. In this case the only design choice remaining was the size of the hidden layer. A 10-hidden-neuron ($N_2=10$) MLP-1 gave good results; no improvement was obtained with $N_2=15$. As a result, all the work described was performed with a (42:10:13) MLP-1 network. The fact that $N_2<N_3$ implies that the output data can be linearly encoded into a reduced number of linearly interdependent variables.

The non-linearity chosen for the hidden layer was the anti-symmetric sigmoid :

$$S(x) = \frac{2}{1 + e^{-x}} - 1 = \tanh\left(\frac{x}{2}\right)$$

The derivative is as simply calculated as in the case of the more usual [0, 1] sigmoid ${}^1/_2\ (1+\tanh({}^x/_2))$:

$$\frac{d\, S(x)}{d\, x} = \frac{1}{2}\left(1 - S(x)^2\right)$$

The input layer and the hidden layer both have offset bias terms appended to their data vectors, feeding the next layer, Fig. 2.

Although it is plausible that the network could have been configured partially connected to give similar results, this was not attempted. Both the forward pass and back-propagation are computationally efficient off-line and in real-time if the matrices are complete, even if sparse. It would only have been useful to partially connect $W^{12}$ and $W^{23}$ if a significant fraction of the elements of these matrices were set to zero. This was not felt to be the case for our problem. In a real implementation, segmentation will always be a practical problem unless the blocks of non-zero weights are compact and few in number.

## III. LEARNING

The starting point for our learning is the conventional back-propagation algorithm, including a momentum term :

$$\delta W^{(k+1)} = -\alpha \frac{\partial(Q)}{\partial W} + \eta\, \delta W^{(k)} \;; \quad Q = \langle error^2 \rangle \qquad (3.1)$$

where W is any weight, $\alpha$ is the Gradient Descent step-length and $\eta$ is the so called momentum term. Q is averaged over both the samples and the outputs.

Our first experience with this convergence technique was frustrating due to the inefficiency of the Gradient Descent technique as implemented in [8]. As most researchers have experienced, we were obliged to play hit-and-miss with the two learning parameters, step-length and momentum, re-learning the optimal parameters each time a new problem was started. The learning was manually interrupted when visibly inefficient, and new parameters were tried. This was repeated until successful, and was annoyingly labour intensive.

We considered this procedure to be far too tedious for attacking many different problems at a time, and we were forced to develop a General Adaptive Recipe (GAR) which converged faster, and most importantly, needed no interactive help and no setting-up decisions. We use the word recipe for this method, since the algorithm used remains one of gradient descent.

The gradient descent was always performed with a batch update, calculating Q for all outputs and all examples, and updating according to Eq. 3.1. Other authors have prescribed updating on the basis of each example, and there is always the compromise of updating on subsets of the training examples. We chose the full batch update since it has a uniquely defined noiseless surface to minimise along. This was essential for using the adapting and resetting to be described later. The danger with example-by-example updating or partial batch updating is that we do not know whether any increase or decrease in Q is global for the whole data set or not, and the adaptation is then dubious. The partial subset update appeared beneficial at times, but led to oscillations in Q for many of the cases tested. We found no robust subset scheduling which needed no user intervention and we remained with the full example set updating.

In order to develop the General Adaptive Recipe, we investigated the following topics :

- Initial Seeding Width
- Initial Seeding Distribution
- Bias Terms
- Learning Rate
- Momentum Term

- 2nd Order Damping
- Output Offset Dead-reckoning
- Output Matrix Dead-reckoning
- Other Methods
- Problem Generality

These points will be discussed in turn, using the nomenclature of Fig. 2. Although many authors have made steps in some of these directions, we have not found them discussed on a single non-trivial problem, nor have we seen the same compromise chosen. For these reasons, we partly overlap with other sources. The GAR is summarized at the end of this section and the choice of parameters is verified for two problems.

*III.1 Initial Seeding Width and Distribution*

Conventionally the weight matrices are seeded with a given width $|W^{12}|$ and $|W^{23}|$ and a flat random distribution. The widths should hopefully not be too critical as the converged solution must be able to "forget" the seeding distribution. A few points concerning this seeding are of interest, however. Permuting the hidden layer neurons, and of course the weights they are attached to, gives us $N_2!$ identical solutions, in so far as the outputs are undisturbed. The anti-symmetric sigmoid (or the symmetric sigmoid with one more change) allows us to invert the signs of $W_{ij}{}^{12}$ and $W^{23}{}_{jk}$ for all i,k and for a given j, again leaving the output undisturbed. This gives $2^{N_2}$ reflection equivalences for each permutation equivalence. We therefore have a total of ($2^{N_2} \times N_2!$) sets of matrices [$\underline{W}^{12}$, $\underline{W}^{23}$] which provide precisely the same given analytical map. For a medium-sized problem such as ours with 10 - 15 neurons in the hidden layer, we have 3.6 $10^9$ - 4.3 $10^{16}$ equivalent minima. For a modest 40 hidden neurons we have $10^{60}$ equivalent minima. This very large number of exchange minima is still relatively small compared with the number of free parameters, 573 in our case and many thousands in the larger case. We can, however, imagine that our initial seeding choice could take us nearer to or further from possible concentrations of these minima.

To begin with, let us consider $N_1$ input neurons. Assuming that the input examples are randomly distributed inside the normalised range [-1, 1] (denoted R[-1, 1]), the hidden layer input distribution will be equal to

$$\text{HIDDEN\_INPUT} = R[-1,\ 1] \times R[-1,1] \times |W^{12}|$$

which has a range $[-N_1\ |W^{12}|,\ N_1\ |W^{12}|]$ and a width $|W^{12}|\sqrt{N_1}$; the ratio of width/range therefore varies as $1/\sqrt{N_1}$. Large input layers will produce small relative widths and large ranges at the hidden layer input. Hidden layer neurons with large inputs are hard on or hard off and will learn very slowly (if at all). We consider it advisable to start with a seeding such that the hidden layer inputs are distributed independently of the number of inputs, and we therefore reduce the range by a factor of $1/N_1$. We also remove the $N_1$-dependent peaking in a simple way by using a random probability function $P(W)=|W|$ obtained by distorting the flat distribution :

$$\frac{W_{12}}{|W_{12}|} = y / \sqrt{|y|} \qquad y = R[-1,\ 1]$$

Suitable choices for $|W^{12}|$ and $|W^{23}|$ are found empirically to be $1.0/\sqrt{N_1+1}$ and $3.0\ /\ \sqrt{N_2+1}$ respectively. This seeding recipe gives hidden and output layer initial distributions which are independent of both $N_1$ and $N_2$, verified with Monte-Carlo tests.

### *III.2 Bias terms*

A large bias term will provoke a large back-propagation update to its dependent weights and a very small bias term will provoke only a very small change. Both extremes must lead to a loss of convergence speed. Since the hidden and output layer input distributions are now independent of the network size, we should be able to choose a suitable bias term recipe for all network sizes. We choose a value which only slightly perturbs the input distribution again tested using Monte-Carlo data. The chosen recipe is

$$\text{Input layer bias (input to hidden layer)} = 0.13\,\sqrt{N_1+1}$$
$$\text{Hidden layer bias (input to output layer)} = 0.04\,\sqrt{N_2+1}$$

In fact the output layer input bias is not used in the final form of the GAR, see Section III.4. The multiplication by the network size factor cancels out the chosen weight distribution width.

### *III.3 Learning Rate and Momentum Coefficients*

The learning rate $\alpha$ determines the change in weights, Eq. 3.1. As previous authors, we found that high initial values, low intermediate values and higher final values were needed as the convergence progressed. No best choice of $\alpha$ can be defined, and we automated an adaptive step-length control based on the improvement to Q for each iteration. The simple recipe is as follows:

$$\alpha^{(k+1)} = \alpha^{(k)}\,(1 + a_{up}) \quad \text{if Q reduced} \qquad (a_{up} > 0)$$
$$\alpha^{(k+1)} = \alpha^{(k)}\,(1+a_{down}) \quad \text{if Q worsened} \qquad (a_{down} < 0)$$

The momentum term $\eta$ defines a first order smoothing filter on the series of weight updates, Eq. 3.1. Simply reducing $\alpha$ in the case of a worsened value of Q is therefore inadequate. We must also forget the previous update by resetting the memory:

$$\delta W^{(k-1)} \rightarrow 0$$

This combination of forgetting and updating has also been developed by Vogl et al. [13]. However, they differ significantly in their use of the momentum term. We have found that a large momentum ~1 is essential in combination with the adaptive step-length. In fact we propose the use of $\eta$=0.999 which provides a low-pass filter of a 1000-iteration bandwidth. Since we usually obtain a reset at least at this rate, the momentum term is almost a perfect step-length integrator in practice.

Now that the learning rate is adaptive, we can choose any starting value for $\alpha$, and always begin with 1.0. The values of $a_{up}$ and $a_{down}$ were chosen to be 0.005 and -0.3 respectively.

We have explored the use of different step-lengths for the $W^{12}$ and $W^{23}$ matrix updates. We have never convinced ourselves that any improvements seen were general, and have no reason to separately choose $\alpha^{12}$ and $\alpha^{23}$.

### *III.4 Output offset dead reckoning*

When the patterns are presented at the output, each output neuron will have an average offset error. Back-propagation propagates this average error back to all weights, whereas it can obviously be fully compensated by the output layer bias term alone. We then only update $W^{12}$ and $W^{23}$ for the non-constant output errors. This output offset dead-reckoning correction can be calculated directly:

$$W^{23}_{bias,j} = W^{23}_{bias,j} - \frac{\langle offset \rangle}{bias} DC$$

The value of the bias term on the output layer now becomes of no significance, as already mentioned. In order to damp this correction we work with DC<1, choosing DC=0.95. We use no momentum on the output layer input bias for obvious reasons. This part of the GAR has its greatest effect during the initial phase of learning.

*III.5 $2^{nd}$ order damping*

Just as the momentum term provides a first order weight update filter, we can add an arbitrary second order filter :

$$\delta W^{(k+1)} = -\alpha \frac{\partial Q}{\partial W} + \eta\, \delta W^{(k)} + \gamma\, [\delta W^{(k)} - \delta W^{(k-1)}]$$

We can choose $\gamma$ to give us a critical damping, corresponding to parabolic extrapolation to the minimum using the curvature $(\delta W^{(k)}-\delta W^{(k-1)})$. During testing we found no systematic improvements using this term $\gamma$, although local improvements were frequently found. If $\gamma$ is large enough to be useful, the convergence ends up worsening at some point, resetting the more useful momentum term. We therefore use $\gamma$=0.

*III.6 Output layer dead-reckoning*

Back propagation optimises in the space of [$W^{12}$, $W^{23}$]. Since the output neuron is linear, the hidden-layer outputs are simply linearly mapped onto the output layer by the matrix $W^{23}$. This linear mapping has a well defined least-squares solution, namely the pseudo-inverse of the equation:

$$W^{23} \, . \, \text{hidden\_output} = \text{final\_output}$$

In this case we only need to minimise the network in a smaller space [$W^{12}$ ], each $W^{12}$ matrix having a defined optimal $W^{23}$. The pseudo-inverse was calculated by Singular Value Decomposition, together with a certain limit on the condition number of the singular values. We are actually generalising the output offset dead-reckoning of the previous section.

It was surprising to us, and still somewhat troubling, that the Output Layer Dead-Reckoning has not proven to be a success, since its simplicity is attractive. We found in practice that the matrix $W^{23}$ found by the pseudo-inverse depended too strongly on how the inverse was conditioned. It gave results which were extremely encouraging for some convergences, but even failed on others, simply stagnating.

We decided that robustness was more vital than occasional high speed and have not managed to tame this method. It is our opinion that the approach ought to yield an advantage, possibly even by alternating methods, but who chooses when to alternate ...? We must also remember that the use of a pseudo-inverse totally destroys the simple parallel nature of the back-propagation algorithm.

*III.7 Other methods*

We have implemented the variation of the step length adaptation proposed as the "Bold Driver" [14] and found it to be about 100 times slower than our General Adaptive Recipe, Fig. 4. This is probably due to their large upward increment, $a_{up}$ = 0.1 instead of 0.005, and no momentum term.

### *III.8 Generality of the Recipe*

The pitfall of a supra-optimisation, looking for the best optimisation recipe, is to optimise the convergence for one particular problem. We have tested the General Adaptive Recipe on many problems, of small and medium size (up to 2000 weights). In six months of use, we have only modified one parameter, the bias, by a factor of 0.3, leading to only 20% better convergence. It is our opinion, therefore, that the recipe proposed is justifiably "General" for this type and scale of problem.

### *III.9 Parameter Sensitivity and Robustness*

We look for two properties of the recipe and its chosen parameters:

- fast convergence
- no convergence stagnation

Figure 5 shows the convergence plot of a typical problem, with $N_1$=9, $N_2$=10, $N_3$=3, 500 examples. Figure 5(a) shows the GAR convergence with 6 different randomly generated weight matrices to begin with. 6 cases converge with similar efficiency. Figure 5(b) shows 10 convergences with fixed learning parameters ($\alpha$, $\eta$) in the range $\alpha$ = 0 0.01 - 0.6 and $\eta$ = 0.7 - 0.9. The poor parameter choices never converge, and the best choices converge reasonably, once they have been identified. The best standard non-adaptive back-propagation did very well, but its parameters had to be found by trial and error. The GAR is therefore efficient.

In order to illustrate the supra-optimisation, we performed single parameter scans on two different problems. Problem A labelled "EXP1" was an $N_1$=5, $N_2$=5, $N_3$=2, 10 examples network and problem B labelled "DIII" was an $N_1$=42, $N_2$=10, $N_3$=13, 500 examples network. Figure 6 (a, b) shows the effect of varying $\eta$; Fig. 7 (a, b) shows the effect of varying the hidden layer input bias; Fig. 8 (a, b) shows the effect of varying the adaptivity $a_{up}$. It is encouraging to see that the supra-optimisation is valid for two very different size problems with 42 and 573 weights respectively.

### *III.10 General Adaptive Recipe*

We summarize the General Adaptive Recipe as follows :

Seeding $W^{12} = 1.0\ (N_1+1)^{-1/2} \times R[-1, 1] \ / \ |R[-1, 1]|^{1/2}$

Seeding $W^{23} = 3.0\ (N_2+1)^{-1/2} \times R[-1, 1] \ / \ |R[-1, 1]|^{1/2}$

Hidden-layer Input Bias = $0.13\ (N_1+1)^{1/2}$

Positive update $a_{up} = 0.005$

Negative update $a_{down} = -0.3$

Initial Learning Rate $\alpha = 1.0$

Momentum $\eta = 0.999$

On the purely practical side, the GAR has been implemented under IDL [15] in an interactive graphical environment providing matrix arithmetic faster than FORTRAN. The GAR procedure is summarized in Appendix A for reference.

## IV. RESULTS

In this section, we present the results for two tokamak equilibrium problems, firstly an illustrative test-case, and secondly with real experimental data. We consider the test-case of an unshaped plasma immersed in a purely vertical field, within a 3:1 rectangular aperture, see Fig. 9. In the large aspect ratio approximation, the poloidal flux distribution is known to be given by

$$\psi(R,Z,\Lambda) = \frac{\mu_0 I_p R}{2\pi}\left[\ln\,(8R/\rho)-2\right] - \frac{\mu_0 I_p}{4\pi}\left[\ln\,(\rho/a) + (\Lambda+0.5)(1-a^2/\rho^2)\right]\rho\cos\omega' \tag{4.1}$$

where $\rho$, a, $\omega'$ are shown in the figure. The free variables are the major radius (R), vertical position (Z) and poloidal asymmetry factor ($\Lambda$), the plasma current $I_p$ being kept constant.

The detection signals are obtained from a set of flux loops placed slightly outside the aperture, shown in the figure, and the inputs to the MLP were the differences in flux with respect to one of the loops. This test-case has the required property of being non-linear by virtue of Eq. (4.1), as well as being simple and descriptive of the full problem.

A set of 1000 equilibria was generated, of which 500 were to be mapped, and the remaining 500 were used for testing the quality of the approximated mapping. The values of [R, Z, $\Lambda$] were varied randomly and uniformly between [0.8 - 0.9, −0.48 - 0.48, 0.5 - 4.0] respectively. The first 500 examples were mapped by both a unique linear relationship obtained by standard SVD techniques and by an MLP-1 with $N_2$ = 10. The quality of the representation was taken to be the mean square residual at the output, normalised to the full range of each output parameter. This exercise was carried out for a varying number of flux-loops, obtaining the results shown in Fig. 10.

The linear representation of $\psi \rightarrow [R,Z,\Lambda]$ was good for very large numbers of flux-loops. However, for smaller numbers of loops, fewer than 15, the representation was inadequate. The MLP-1 mapping maintained low residuals even when the total number of flux loops was decreased to as few as 6. An example of the quality of representation, fitted value vs. actual value, is shown in Fig. 11 for this case with only 6 input signals and 5 flux-differences. The MLP-1 clearly is able to use its inherent non-linearity to solve the type of non-linear mapping problem represented by Eq. (4.1).

The next step was to make the problem more difficult by increasing the range of R to [0.7 - 0.9], so that either the inner or outer wall defined the effective aperture. This produces a severe bend in the mapping when the plasma surface touches both walls. Figure 12 shows the flux differences for 4 of 9 flux loops as the major radius is varied over the full range, for a vertically centred and fixed $\Lambda$ plasma, illustrating the structure of the mapping required. This problem with a gradient discontinuity was solved using an ($N_1 = 9$, $N_2 = 10$, $N_3 = 3$) MLP-1 network with a precision of 1.2 % FS. The crosses in Fig. 12 show the fitted data and the solid line shows the underlying function.

For the second problem a DIII-D tokamak experimental database was used. Since the equilibria are experimental, both **M** and **G** now have intrinsic noise. During the MLP training, we again used a random set of 500 equilibria as the training set, and the quality of the fit was subsequently tested on 500 different equilibria.

The input data for training were a combination of 20 magnetic flux loop differences and 22 poloidal magnetic field probes, giving a total of 42 inputs. The outputs were chosen as 13 geometric variables, listed in Table

I, some of which are illustrated in Fig. 1. We chose $N_2$=10 hidden layer neurons, and obtained very good residuals, listed in Table I. Increasing the size of $N_2$ made no improvement to the residual. Table I also shows the residuals from a linear mapping as well as the physical range of each parameter, and the minimum and maximum absolute errors. Using the MLP-1 mapping we generally obtain an improvement of 30-50 % in the residuals compared with a general linearisation using all of the 42 input parameters. The parameter which is least well fitted is the top triangularity ($\sigma_j$ = 5.8% FS). The MLP-1 fitting attempted to minimise this particular $\sigma_j$, and this was done at the expense of, for example, the vertical position of the plasma current ($\sigma_j$ = 0.9% FS). In the linearised case, the output variables are fitted independently and weighting them differently does not change the solution. In the MLP-1 case, we can reduce $< \sigma_j^2 >$ by worsening the fit on one output variable to benefit any less well fitted variables. For the work described, the $\sigma_j$ were simply weighted inversely to their full-scale range although such a global mapping might benefit from a selected output parameter weighting, such as the inverse residual of the linearised fit.

Figure 13 shows the residuals for four of the standard control parameters: top gap, inner gap, geometric centre and X-point Z-position and compares the MLP-1 fit with the DIII-D adhoc trial function fit [1]. These results together indicate that the MLP-1 is a good tool for providing the mapping function $\hat{G}(\mathbf{M})$ for our particular problem.

Figure 14 presents a method of visualising the use of the sigmoid function which provides the necessary non-linearity. The hidden layer inputs and outputs for all the trained examples are plotted for all 20 hidden layer neurons of a converged problem. Some neurons operate near the origin and are therefore almost linear. Others operate near the maximum curvature at |input|=1.314. Others cover a range of both positive and negative curvature. Some neurons cover a small range of input, and others a large range. This Sigmoid Distribution Plot was used when pruning a network to remove the least useful neurons before relearning. An objective criterion of usefulness was found to be the standard deviation of the sigmoid curvature over the learning set. This criterion always corresponded to the interactive visual choice made when pruning the network by hand.

## V. HARDWARE IMPLEMENTATION

Hardware has already been constructed to provide a piecewise linear approximation for the mapping $\hat{G}$ for the TCV tokamak being built at the CRPP. This system is a modular matrix multiplier using DAC-multipliers and is described in detail elsewhere [6, 16]. The system for the TCV tokamak will have a 128 x 24 matrix and two 24 x 24 matrices, giving a total of 4224 matrix weights. A library of weights is stored in the matrix multiplier and any one can be called up within 1.3 msec. The whole system is controlled using the BITBUS fieldbus. The bandwidth of the DAC-multipliers is very high, retaining 8 bits of resolution at over 50 kHz. The corresponding number of FLOPS to obtain this throughput of continuous digital matrix multiplications is of the order of 900 MFLOPS. Figure 15 shows a schema of the matrix modularity.

Since the $W^{12}$ and $W^{23}$ matrices can be constructed using this existing hardware, and an analog sigmoid is easily reproduced, Section VIII, the system can be reconfigured from a piecewise linear mapping to an MLP-1 mapping by simply recabling the modules. In this way we will obtain a fast, fairly large and very precise MLP-1 mapping using available technology.

## VI. DISCRETISATION

If the matrix multiplications in the MLP forward pass are to be carried out by some form of hybrid analog-digital system using multiplying DAC's, pulse coded multiply etc., the weights will have to be stored in a digital form. The question which then arises is to what extent can the continuous valued weights can be discretised without loss of output precision. We need to determine the required digital resolution.

The MLP-1 solution found in Section IV was discretised as follows. The full range of both $W^{12}$ and $W^{23}$ was generously taken as $\pm$ 5.0, Fig. 16. The weights were then discretised with the number of bits varying between 8 and 16. Figure 17 shows that down to 12 bits there is no loss of precision, that is to say no significant increase in $\sigma^2$. As we reduce the number of bits further, $\sigma^2$ increases, rising to 25 times its continuous value with only 8 bit resolution in the weights. Figure 16 also illustrates a similar trend for the linear mapping. The MLP-1 mapping is actually more robust than the linear mapping when we discretise the weights.

This loss of precision might be partly countered by relearning with discretised weights. Since we shall have 12 bit resolution available, this will not be necessary. At present, the only hybrid technique capable of providing this resolution is the multiplying DAC described in Section V. Other techniques are imaginable, but are mostly limited to a smaller number of bits.

## VII. NOISE SENSITIVITY

In a real system there are several forms of noise. The inputs may contain random noise, such as voltage pickup; this noise will be independent of the signal:

$$\text{Signal} = \text{Signal} + N(o,\varepsilon) \times \text{Max } |\text{Signal}| \tag{7.1}$$

where $N(o,\varepsilon)$ is a normal distribution with mean 0 and standard deviation $\varepsilon$. The inputs may have randomly distributed calibration errors; this noise will be proportional to the signal and systematic for each input :

$$\text{Signal} = \text{Signal} \times (1 + N(o, \varepsilon)) \tag{7.2}$$

The weights may also have a random noise due to production tolerances. The weight noise can also be of the same two types mentioned above :

$$\text{Weight} = \text{Weight} + R(0, \varepsilon) \times \text{Max } |\text{Weight}| \tag{7.3}$$

$$\text{Weight} = \text{Weight} \times (1 + R(0, \varepsilon)) \tag{7.4}$$

We independently subjected the solution found in Section IV to these four noise types, and also compared the noise-sensitivity of the linear mapping, Figure 18. The linear mapping is more sensitive to all four noise types, especially the relative noise distributions (a, c). The relative input noise, Fig. 18(a), into the MLP-1 shows a degradation at 1%, corresponding to a reasonable estimate of the calibration accuracy of the inputs. The linear mapping is already very poor with this noise level.

The noisy weights show a similar behaviour, Fig. 18(c, d). The $10^{-3}$ error level corresponds roughly to the 12 bit requirement described in the previous section.

## VIII. SEGMENTED SIGMOID

Many techniques have been proposed for implementing a sigmoid transfer function. Some of these techniques use matched transistor pairs, which may have temperature stability problems. We propose to use a piecewise linearised or segmented sigmoid, with one segment through the origin and three segments plus a saturation on either side, Fig. 19. This will be achieved using the simple circuit shown in Fig. 20.

This segmented sigmoid was tested on the problem solved in Section IV. The mean square residual increased from $1.58\ 10^{-3}$ to $1.8\ 10^{-3}$ without relearning. Relearning could be used to reduce this residual if it were considered necessary.

## IX. SUMMARY

We have considered a real practical problem of the non-linear mapping between tokamak measurements and derived geometric parameters. Although the specific problem is somewhat esoteric, it is one of a large class of similar practical problems, which require a continuous functional map to be derived from a set of examples derived from a known model. The MLP-1 clearly provides a convenient solution to this problem which previously had been approached using a piecewise linear mapping. Furthermore since the latter mapping required large fast matrix calculations to be provided by combined analog-digital hardware, the same hardware could be used, without modification, to generate an MLP-1 mapping. These two considerations led us to look closely at several practical questions concerning the implementation of an MLP-1. The results are most encouraging and suggest that the MLP-1 naturally provides a mapping which is not only more precise and simpler to use than the piecewise linear mapping, but is also more robust. The presence of gradient discontinuities does not appear to be a problem.

In order to complete this study, we developed a General Adaptive Recipe for back-propagation which requires no human interaction when learning MLP mappings, and is much faster than non-adaptive learning. We described the robustness in some detail for two of the many problems we have solved.

## ACKNOWLEDGEMENTS

We are most grateful to the DIII-D physics team, especially Drs John Ferron and Lang Lao for access to the DIII-D database used in this study. We thank Drs Tony Taylor, Chris Bishop and Michael Dutch for their comments on this paper. The applicability of the MLP to this problem was originally discussed with Dr. Robert W. Means. The work was partly supported by the Fonds national suisse de la recherche scientifique.

## REFERENCES

[1] S. Kinoshita, H. Fukumoto, A.G. Kellman et al., General Atomics Report No. GA A 19585, 1989

[2] T.H. Osborne, H. Fukumoto, N. Hosogane et al., Bull. Am. Phys. Soc. Vol. 31(9), pp 1502, 1986

[3] H. Fukumoto, W. Howl, G. Jackson et al., private communication - Shape control workshop, Lausanne, 1987

[4] B.J. Braams, W. Jilge and K. Lackner, Nucl. Fusion Vol. 26, pp. 699, 1986

[5] F. Hofmann and G. Tonetti, Nucl. Fusion Vol. 28, pp. 519, 1988

[6] J.B. Lister, Ph. Marmillod and J.-M. Moret, Lausanne Report LRP 332/87, 1987

[7] J.B. Lister, H. Schnurrenberger, Lausanne Report LRP 397/90, 1990, to be published.

[8] D.E. Rumelhart and J.L. McClelland, Parallel Distributed Processing: Explorations of the Microstructures of Cognition, Vols. I, II and III, MIT Press, 1986

[9] G. Cybenko, Tufts University Report, Medford, U.S.A., 1988

[10] R. Hecht-Nielsen, Proc. INNS Annual Meeting, San Diego, U.S.A., 1988

[11] K. Funahashi, Neural Networks Vol. 2, pp 183, 1988

[12] T. Poggio and F. Girosi, roc. of IE 78 (9), pp 1481, 1990.

[13] T.P. Vogl, J.K. Mangis, A.K. Rigler, W.T. Zink and D.L. Alkon, Biological Cybernetics 59, pp. 257, 1988.

[14] R. Battiti and F. Masulli, Proc. INCC 1990 (Paris), IEEE, pp. 757, 1990.

[15] Interactive Data Language, R.S.I. Inc., Boulder, CA (USA).

## FIGURE CAPTIONS

Fig. 1 A schematic of the DIII-D tokamak, showing the measurements and some of the derived plasma parameters to be mapped to

Fig. 2 A schematic description of the 1-hidden-layer Multi Layer Perceptron (MLP-1) reproducing a pillar

Fig. 3 The basic MLP-1 (2:8:1) and MLP-2 (2:4:1:1) output structures

Fig. 4 Bold Driver recipe compared with the General Adaptive Recipe

Fig. 15 Schema of the hybrid analog digital matrix multiplier to construct $W^{12}$ or $W^{23}$

Fig. 16 Distribution of matrix weights for (a) the MLP-1 mapping showing $W^{12}$ (solid) and $W^{23}$ (dashed) matrices, and (b) the linear mapping

Fig. 17 Effect of the discretisation of the digital matrix weights.

Fig. 18 Effect of different noise sources on the output precision. The abscissa is the value of $\varepsilon$ in Eqs 7.1 - 7.4, (a) Eq. 7.2 for input signal, (b) Eq. 7.1 for input signal, (c) Eq. 7.4 for the matrix weights, (d) Eq. 7.3 for the matrix weights.

Fig. 19 The segmented sigmoid (solid line) and the true sigmoid (dotted line). The error x 100 is shown as a dashed line

Fig. 20 The segmented sigmoid circuit

**Table I**

Ranges and residuals of the approximated plasma parameters using an MLP-1 or linear mapping. The input uses both flux and field probes, the output consists of 13 geometrical parameters (% FS signifies percentage of full scale range).

| Quantity | range | | MLP-1 | | | | Linear | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $\sigma_j$(abs) | $\sigma_j$(%FS) | min(abs) | max(abs) | $\sigma_j$(abs) | $\sigma_j$(%FS) | min(abs) | max(abs) |
| | a | b | | | | | | | | |
| X-point-Z (cm) | -127.2 | -106.8 | 0.216 | 1.1 | -0.804 | 0.909 | 0.350 | 1.7 | -1.889 | 1.627 |
| X-point-R (cm) | 141.6 | 155.2 | 0.150 | 1.1 | -1.114 | 0.796 | 0.188 | 1.4 | -1.298 | 1.680 |
| Top-gap (cm) | 8.03 | 37.67 | 0.449 | 1.5 | -2.161 | 1.848 | 1.003 | 3.4 | -3.451 | 13.452 |
| Inner gap (cm) | 0.61 | 12.08 | 0.096 | 0.8 | -0.503 | 0.340 | 0.142 | 1.2 | -1.025 | 0.625 |
| Current-Z (cm) | 2.09 | 15.60 | 0.117 | 0.9 | -0.523 | 1.420 | 0.042 | 0.3 | -0.124 | 0.380 |
| R-surface (cm) | 166.9 | 173.8 | 0.055 | 0.8 | -0.352 | 0.307 | 0.085 | 1.2 | -0.369 | 1.325 |
| Internal inductance | 0.70 | 1.93 | 0.009 | 0.7 | -0.063 | 0.022 | 0.019 | 1.5 | -0.070 | 0.176 |
| $\beta_p$ | 0.03 | 1.41 | 0.008 | 0.6 | -0.049 | 0.044 | 0.009 | 0.6 | -0.092 | 0.065 |
| K (b/a) | 1.63 | 2.08 | 0.005 | 1.0 | -0.029 | 0.021 | 0.006 | 1.3 | -0.052 | 0.026 |
| Top-triangularity | 0.16 | 0.57 | 0.029 | 5.8 | -0.089 | 0.073 | 0.028 | 6.7 | -0.090 | 0.103 |
| Bottom-triangularity | 0.25 | 0.43 | 0.002 | 1.4 | -0.012 | 0.018 | 0.003 | 1.9 | -0.036 | 0.027 |
| Elongation on axis | 1.15 | 1.91 | 0.007 | 0.9 | -0.084 | 0.036 | 0.012 | 1.6 | -0.139 | 0.109 |
| X-section area ($m^2$) | 1.77 | 2.27 | 0.006 | 1.1 | -0.025 | 0.023 | 0.010 | 2.1 | -0.056 | 0.047 |

**APPENDIX A : The General Adaptive Recipe**

The $N_{ex}$ examples in the range [-1, +1] comprise a matrix

$$\text{EXAMPLES\_INPUT}\ (N_1 + 1,\ N_{ex})$$

where each example has the hidden-layer input bias as the $N_1+1$ input. The input to the hidden layer is the matrix product (#) given by

$$\text{HIDDEN\_INPUT}\ (N_2,\ N_{ex}) = W^{12}(N_2,\ N_1+1)\ \#\ \text{EXAMPLES\_INPUT}$$

The output of the hidden layer is the compressed input, namely

$$\text{HIDDEN\_OUTPUT}\ (N_2,\ N_{ex}) = S(\text{HIDDEN\_INPUT})$$

and the output layer is

$$\text{OUTPUT}\ (N_3,\ N_{ex}) = W^{23}\ (N_3,\ N_2)\ \#\ \text{HIDDEN\_OUTPUT} + \text{OUTPUT\_OFFSET}\ (N_3)$$

The error is given by

$$\text{OUTPUT\_ERROR}\ (N_3,\ N_{ex}) = \text{OUTPUT} - \text{EXAMPLES\_OUTPUT}$$

The output error is first corrected for the offset

$$\text{OUTPUT\_OFFSET}\ (N_3) = \text{OUTPUT\_OFFSET} - \text{DC}\ \langle \text{OUTPUT\_ERROR} \rangle_{N_{ex}}$$

$$\text{OUTPUT\_ERROR} = \text{OUTPUT\_ERROR} - \text{DC}\ \langle \text{OUTPUT\_ERROR} \rangle_{N_{ex}}$$

The mean square error is

$$SUMSQ = \langle\langle OUTPUT\_ERROR^2 \rangle_{N_{ex}} \rangle_{N_3}$$

If the value of SUMSQ has not improved, then we back-off the weight update and reset the learning memory :

$$W^{12} = W^{12} - \delta W^{12}$$

$$W^{23} = W^{23} - \delta W^{23}$$

$$\delta W^{12} = \delta W^{23} = 0$$

Finally, the step length is reduced

$$\alpha = \alpha \, (1 + a_{down})$$

If the value of SUMSQ has improved, we back-propagate the error as follows :

$$\varepsilon W^{23} = \frac{\partial SUMSQ}{\partial W^{23}}$$

$$= OUTPUT\_ERROR \ \# \ (HIDDEN\_OUTPUT)^T$$

HIDDEN_GRADIENT $(N_2, N_3)$

$$= \frac{\partial \ HIDDEN\_OUTPUT}{\partial HIDDEN\_INPUT}$$

$$= \frac{1}{2} \, (1.0 - HIDDEN\_OUTPUT^2)$$

WEIGHTED-ERROR $(N_2, N_{ex})$

$$= \frac{\partial SUMSQ}{\partial HIDDEN\_OUTPUT}$$

$$= (W^{23})^T \ \# \ OUTPUT\_ERROR$$

$$\text{HIDDEN\_INPUT\_ERROR } (N_2, N_{ex}) = \frac{\partial \text{SUMSQ}}{\partial \text{HIDDEN\_INPUT}}$$

$$= \frac{\partial \text{SUMSQ}}{\partial \text{HIDDEN\_OUTPUT}} \cdot$$

$$= \frac{\partial \text{HIDDEN\_OUTPUT}}{\partial \text{HIDDEN\_INPUT}}$$

$$= \text{WEIGHTED\_ERROR \# HIDDEN\_GRADIENT}$$

$$\varepsilon W^{12}(N_2, N_1 + 1) = \frac{\partial \text{SUMSQ}}{\partial W^{12}}$$

$$= \text{HIDDEN\_INPUT\_ERROR \# EXAMPLES\_INPUT}$$

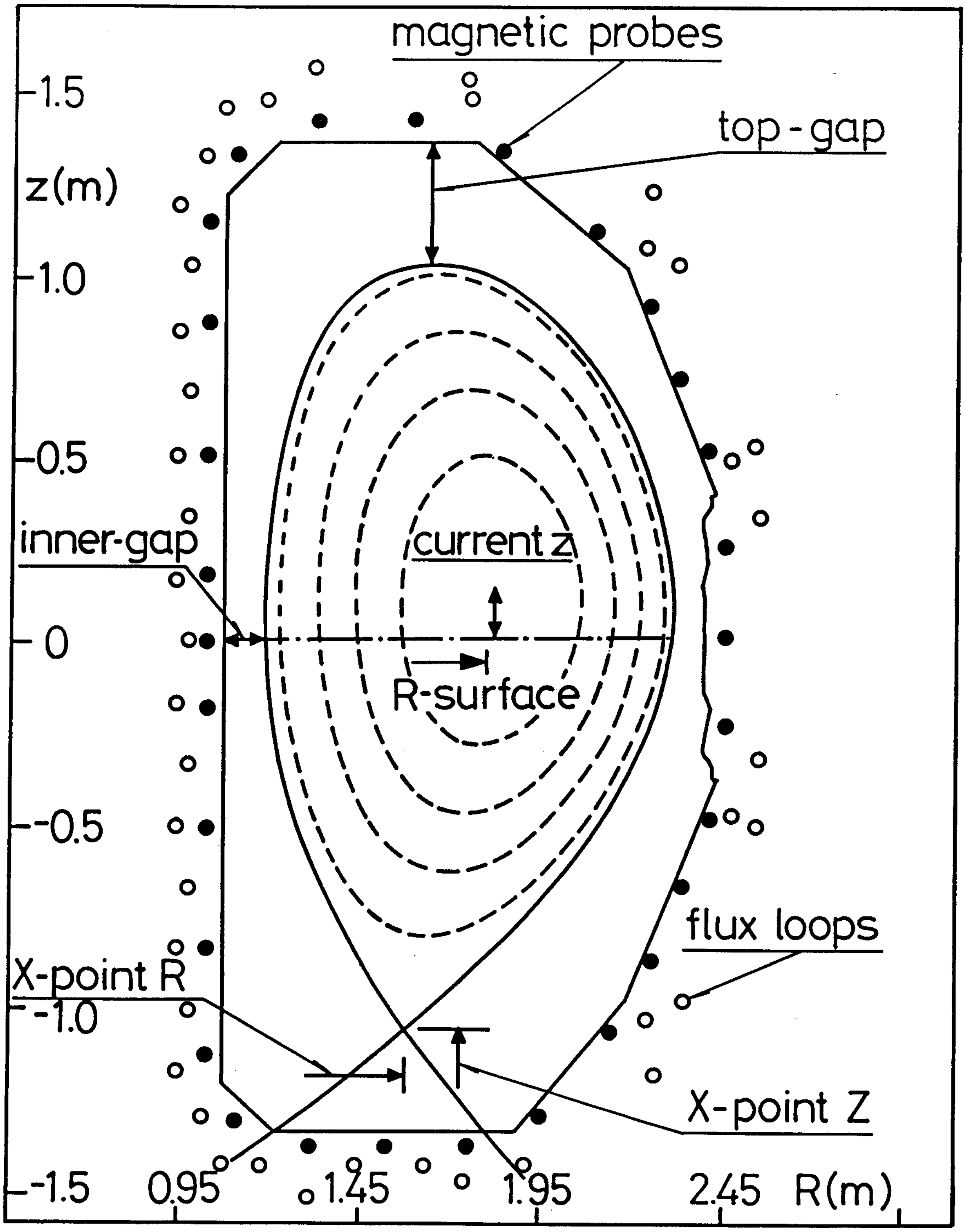
Having obtained the first derivative of SUMSQ with respect to all free parameters, we perform the Gradient Descent :

$$\delta W^{12} = \alpha \varepsilon W^{12} + \eta \delta W_{OLD}{}^{12}$$
$$\delta W^{23} = \alpha \varepsilon W^{23} + \eta \delta W_{OLD}{}^{23}$$
$$W^{12} = W^{12} + \delta W^{12}$$
$$W^{23} = W^{23} + \delta W^{23}$$
$$\delta W_{OLD}{}^{12} = \delta W^{12}$$
$$\delta W_{OLD}{}^{23} = \delta W^{23}$$

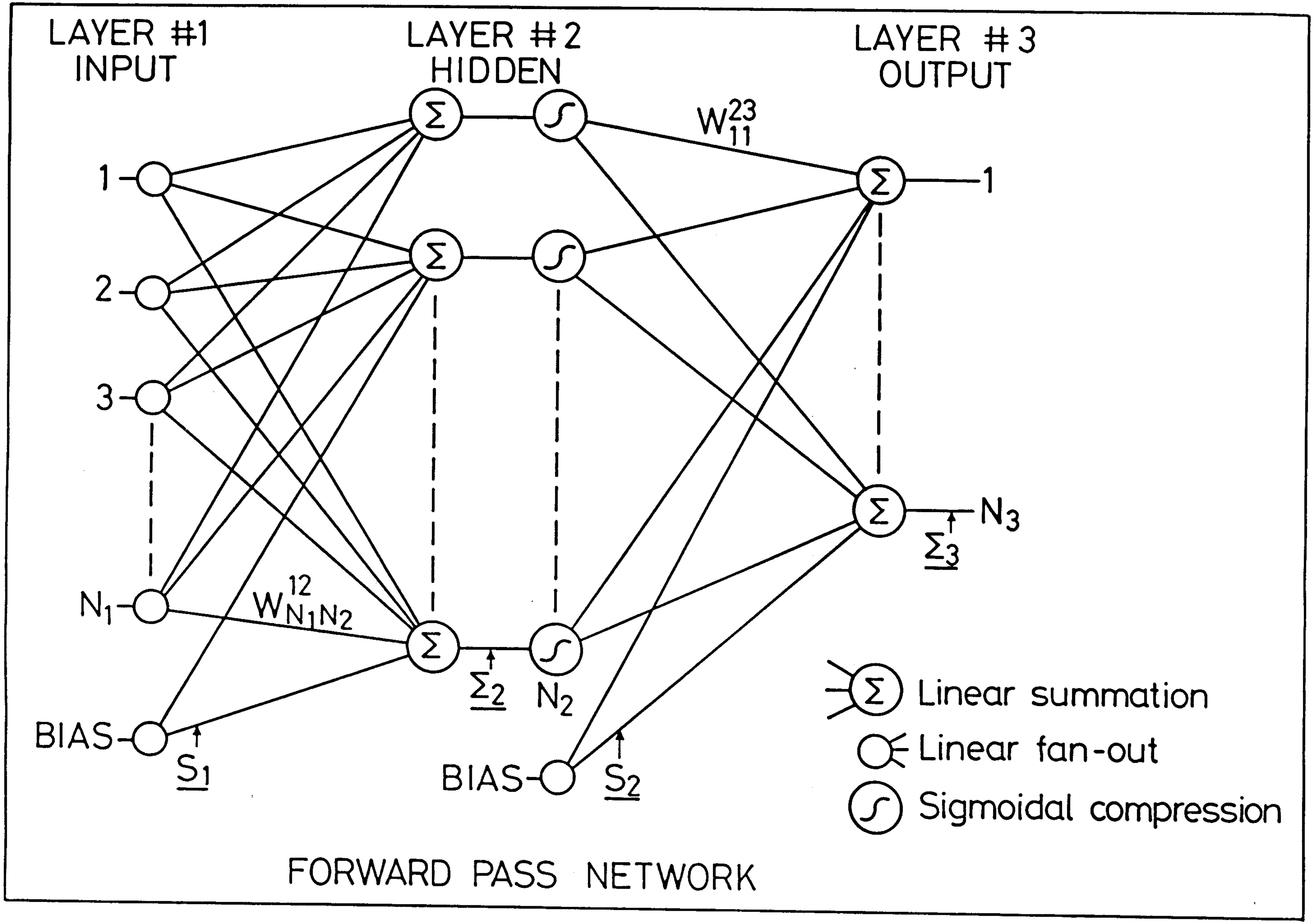
The step length is then adapted upwards :

$$\alpha = \alpha\ (1 + a_{up})$$

Fig. 1



magnetic probes
top-gap
z(m)
1.5
1.0
0.5
0
-0.5
-1.0
-1.5
inner-gap
current z
R-surface
flux loops
X-point R
X-point Z
0.95
1.45
1.95
2.45
R(m)



LAYER #1
INPUT
LAYER #2
HIDDEN
LAYER #3
OUTPUT
1
2
3
$N_1$
BIAS
$W^{12}_{N_1N_2}$
$W^{23}_{11}$
$\underline{S_1}$
$\underline{\Sigma_2}$
$N_2$
BIAS
$\underline{S_2}$
1
$N_3$
$\underline{\Sigma_3}$
Linear summation
Linear fan-out
Sigmoidal compression
FORWARD PASS NETWORK


Fig. 2

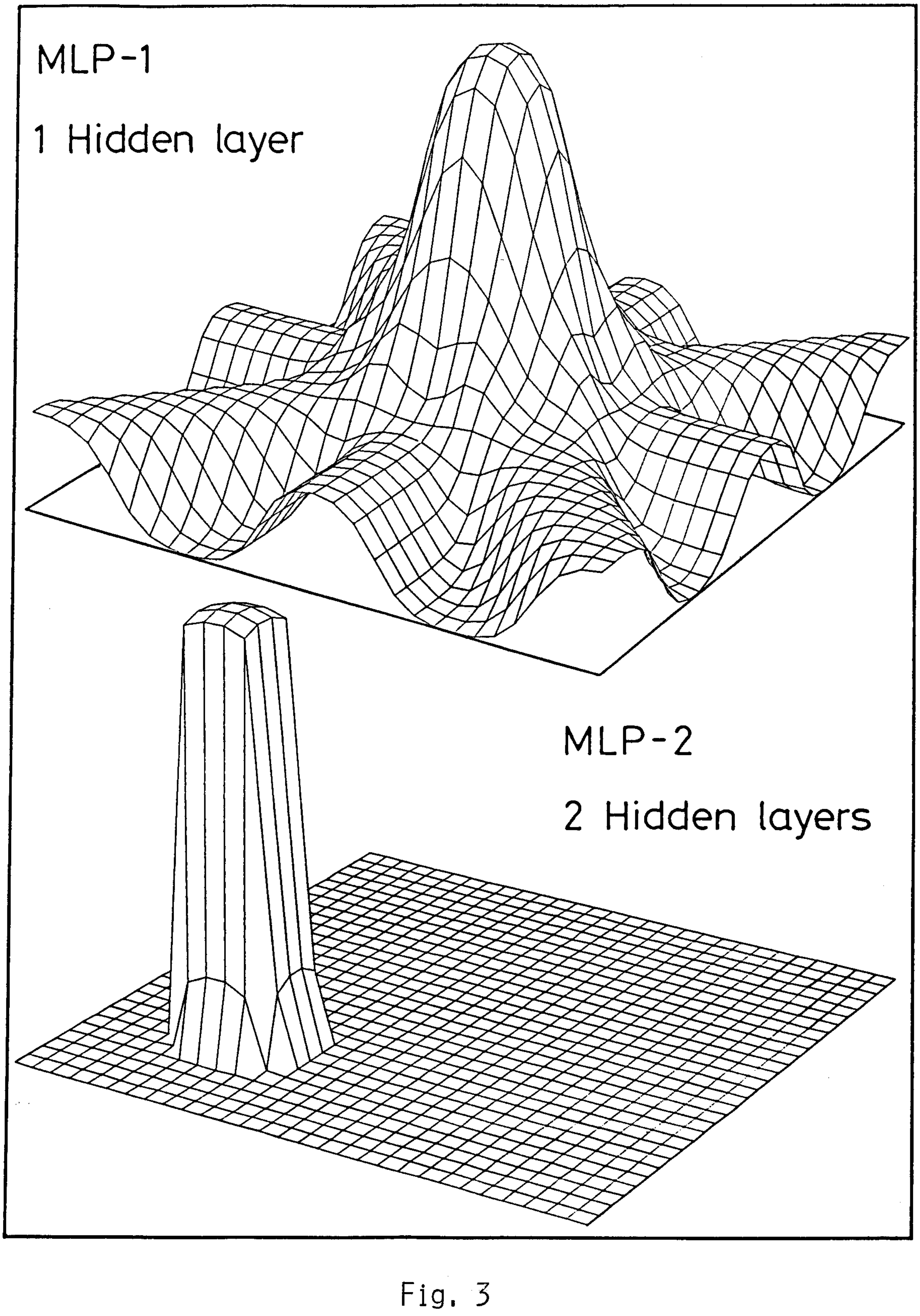



Fig. 3

Fig. 4

BOLD DRIVER

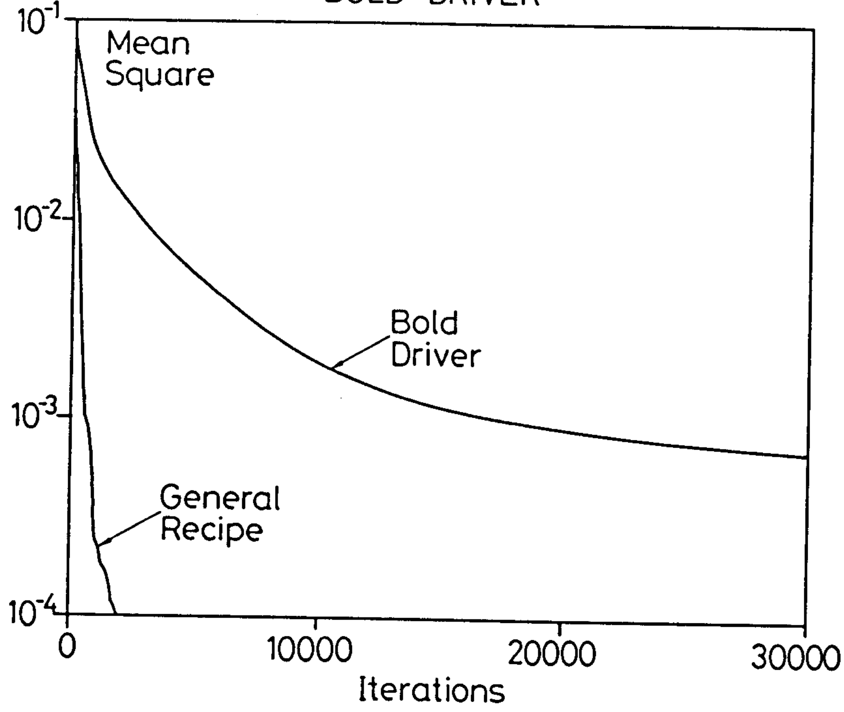
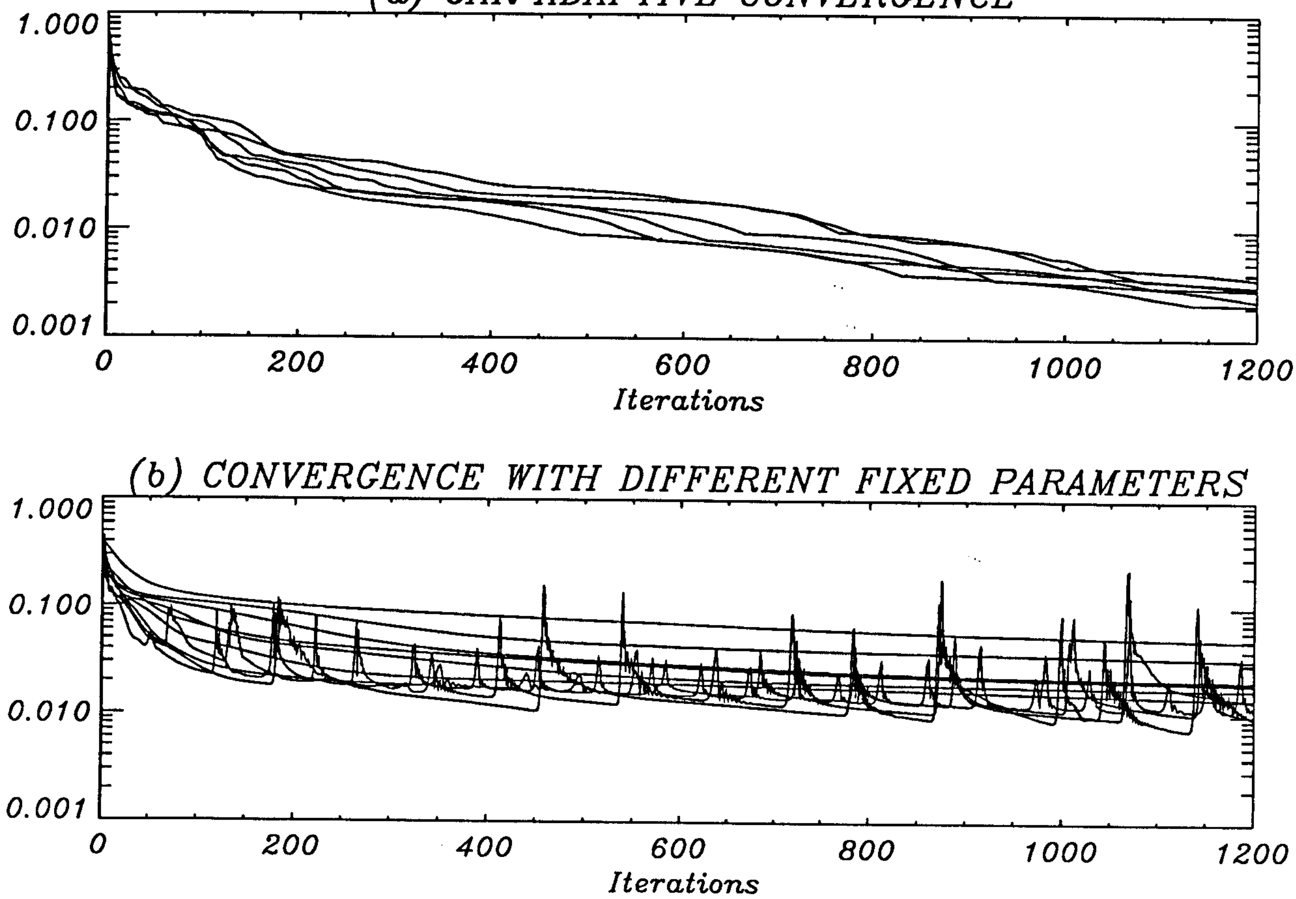
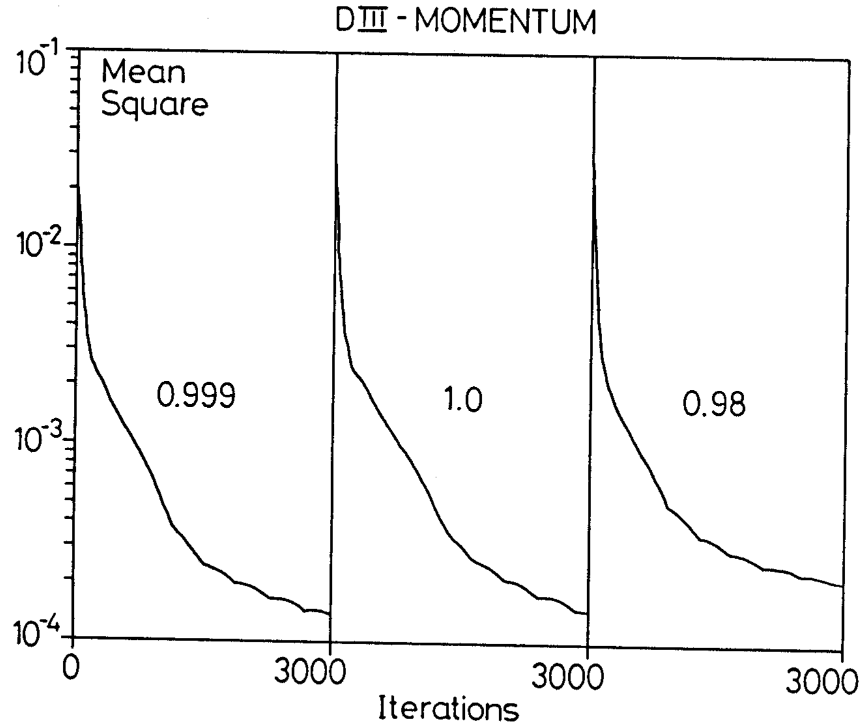



Fig. 5

(a) GAR ADAPTIVE CONVERGENCE



<u>Fig. 6</u>


DIII - MOMENTUM
Mean
Square
10^-1
10^-2
10^-3
10^-4
0.999
1.0
0.98
0
3000
3000
3000
Iterations

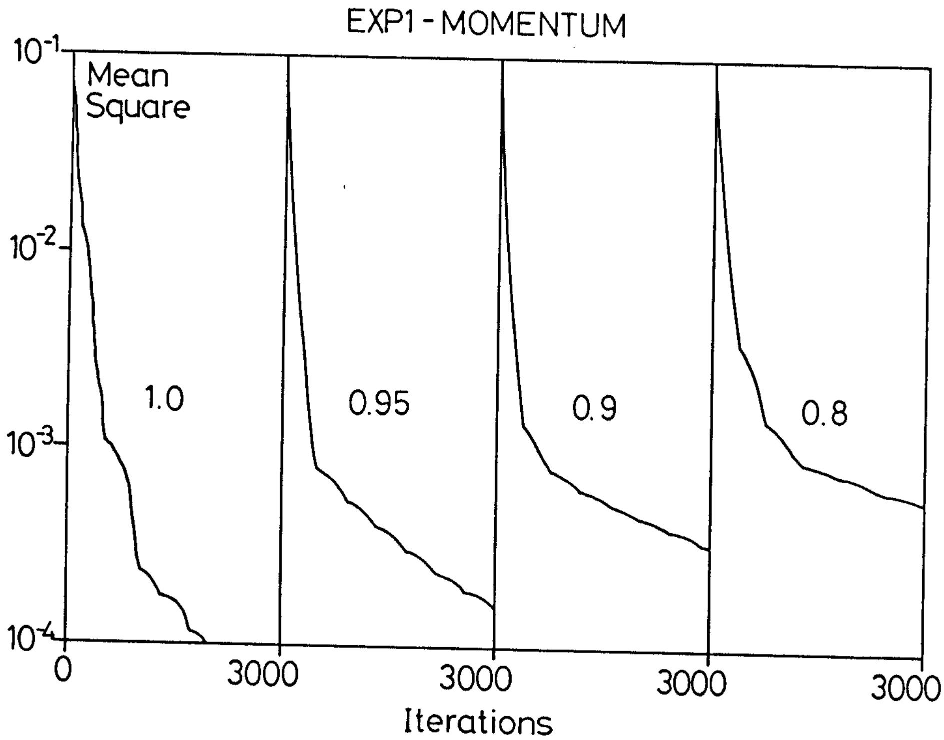
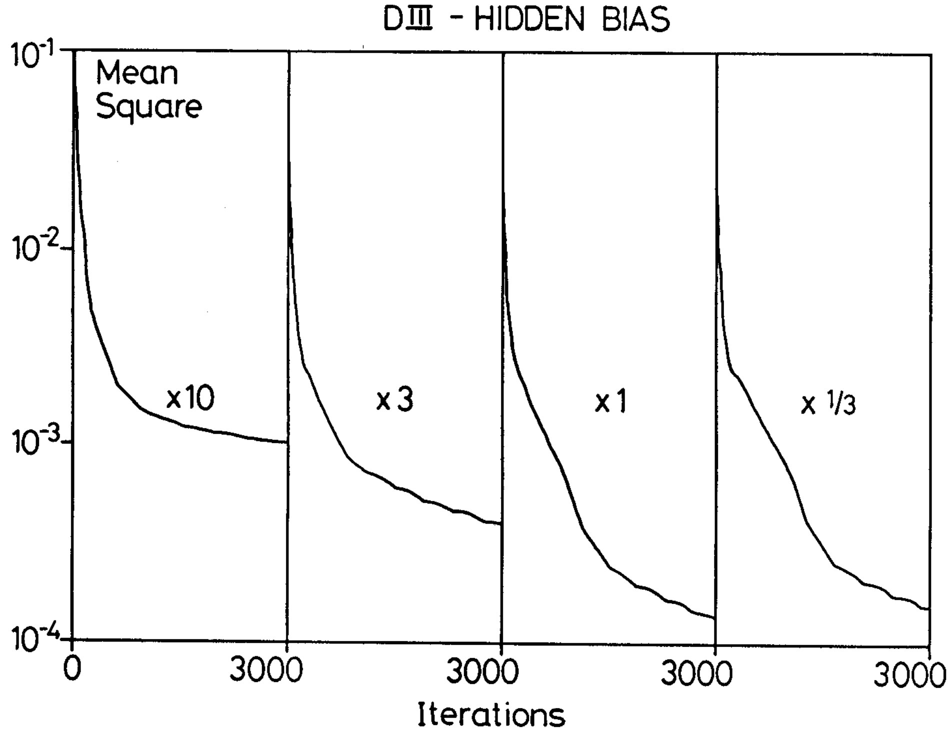

EXP1 - MOMENTUM
Mean
Square
10^-1
10^-2
10^-3
10^-4
1.0
0.95
0.9
0.8
0
3000
3000
3000
3000
Iterations

Fig. 7


DIII - HIDDEN BIAS
Mean Square
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
x10
x3
x1
x 1/3
0
3000
3000
3000
3000
Iterations

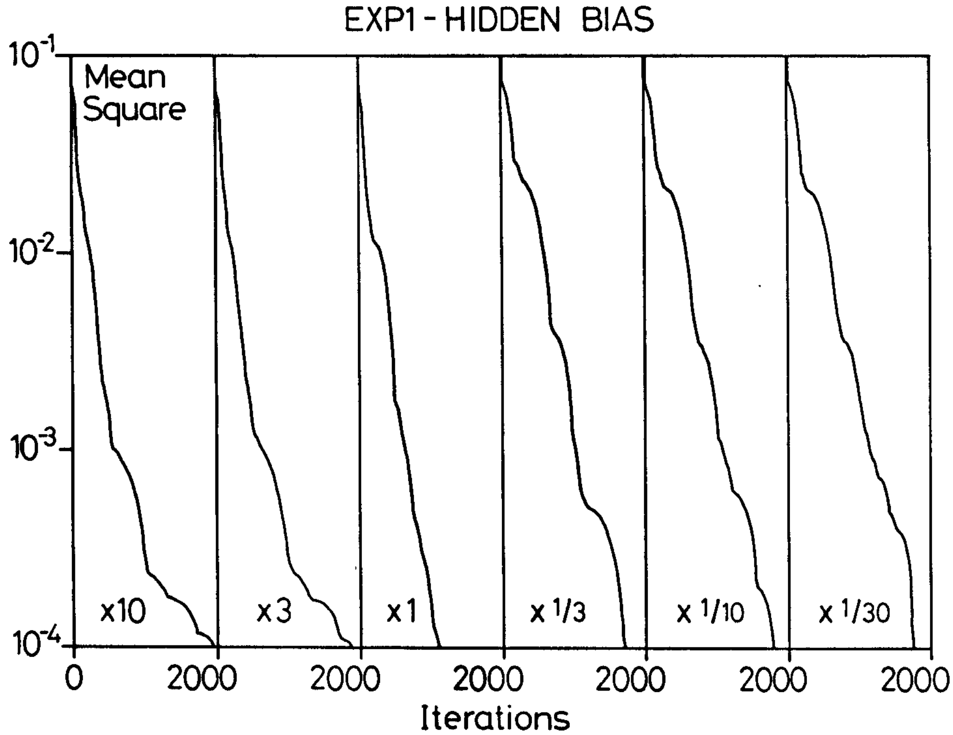
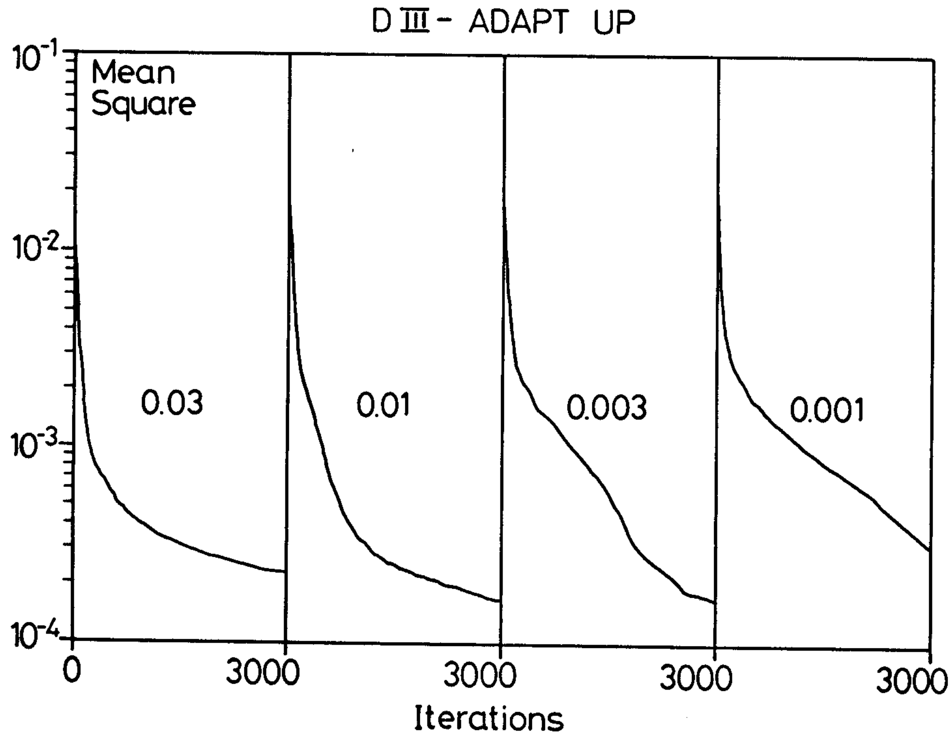

EXP1 - HIDDEN BIAS
Mean Square
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
x10
x3
x1
x 1/3
x 1/10
x 1/30
0
2000
2000
2000
2000
2000
2000
Iterations

Fig. 8


D III - ADAPT UP
Mean Square
10^-1
10^-2
10^-3
10^-4
0.03
0.01
0.003
0.001
0
3000
3000
3000
3000
Iterations

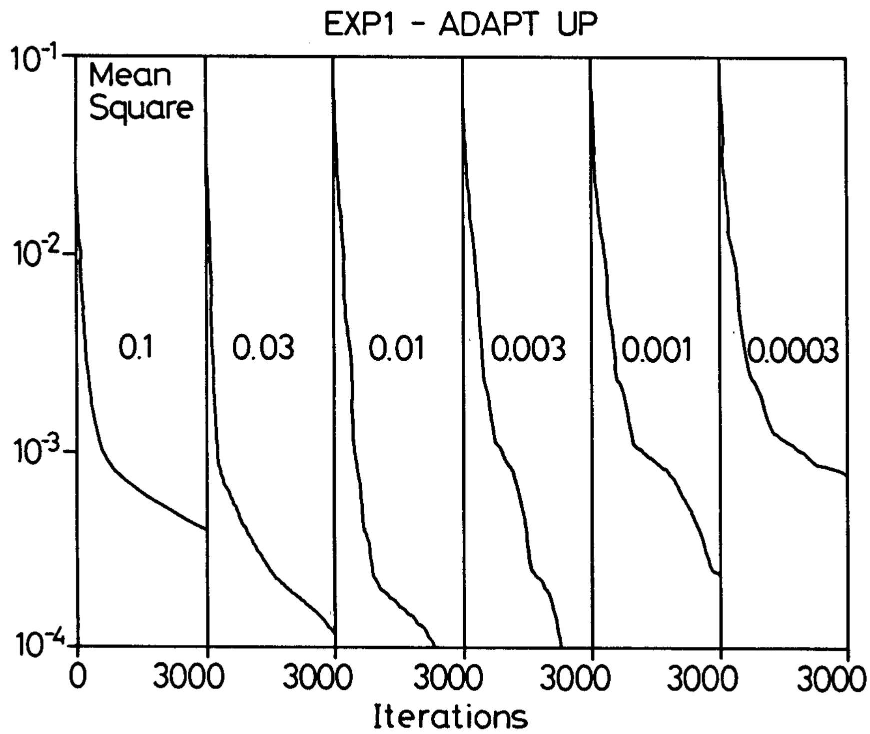

EXP1 - ADAPT UP
Mean Square
10^-1
10^-2
10^-3
10^-4
0.1
0.03
0.01
0.003
0.001
0.0003
0
3000
3000
3000
3000
3000
3000
Iterations

Fig. 9

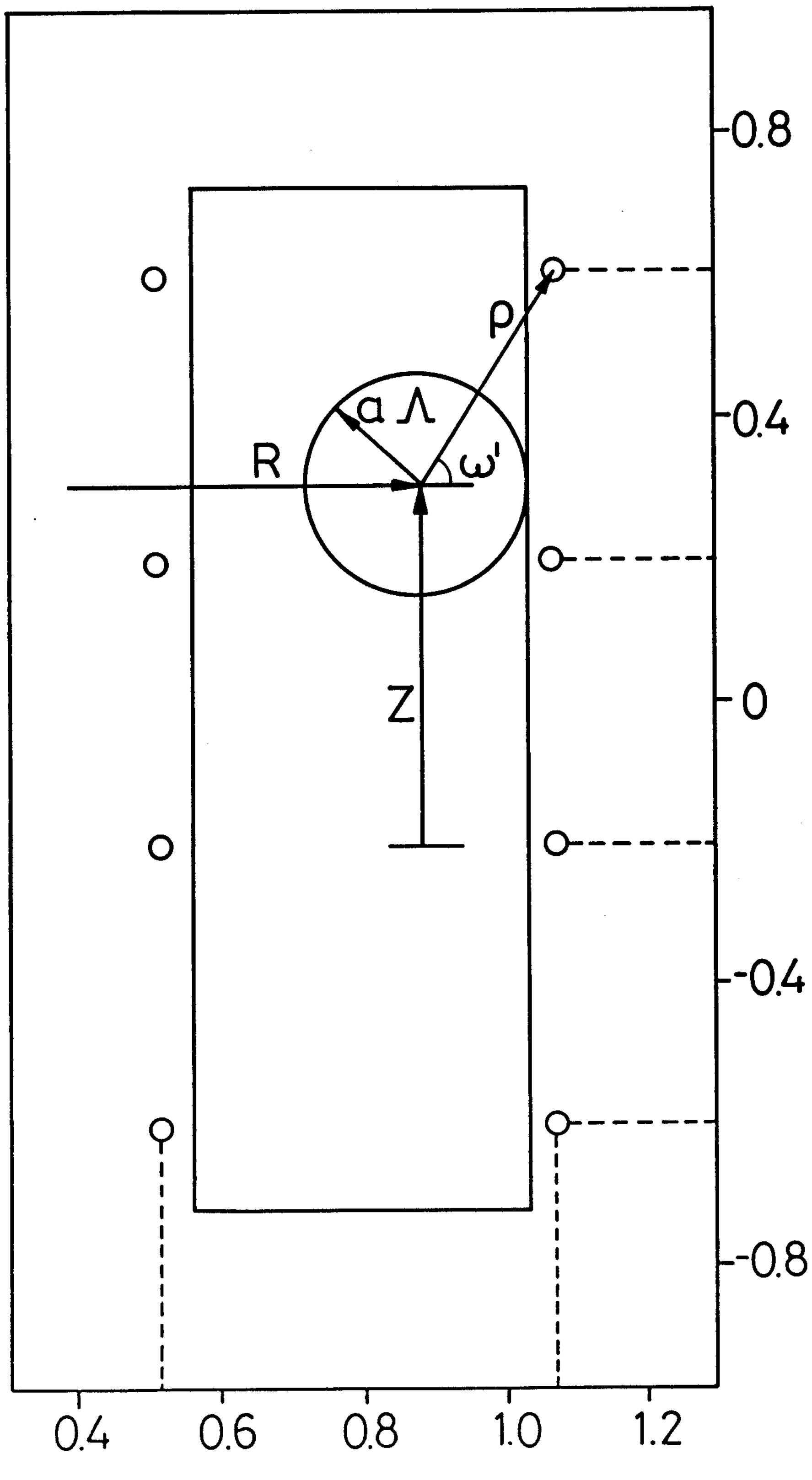
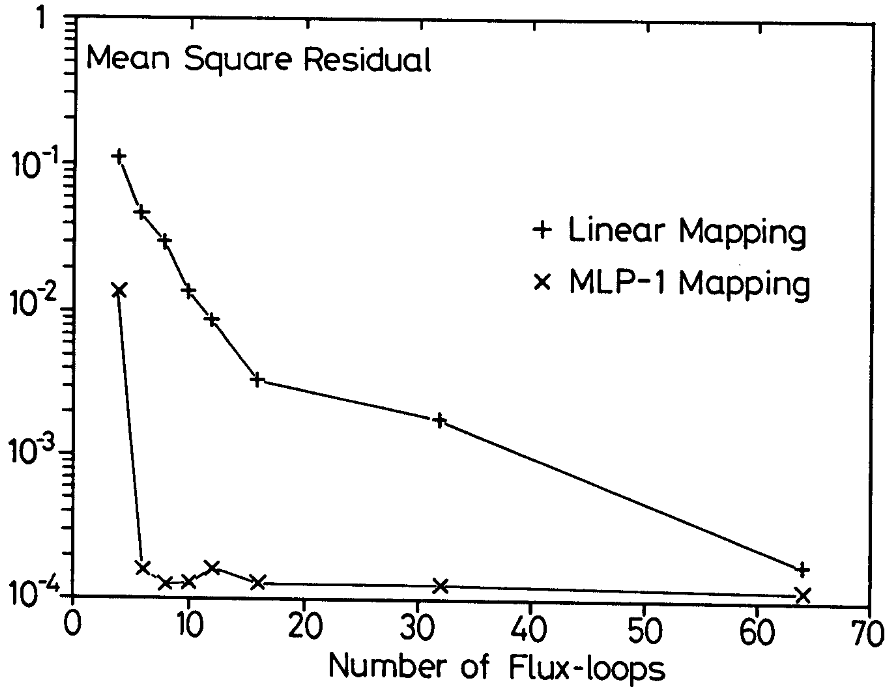
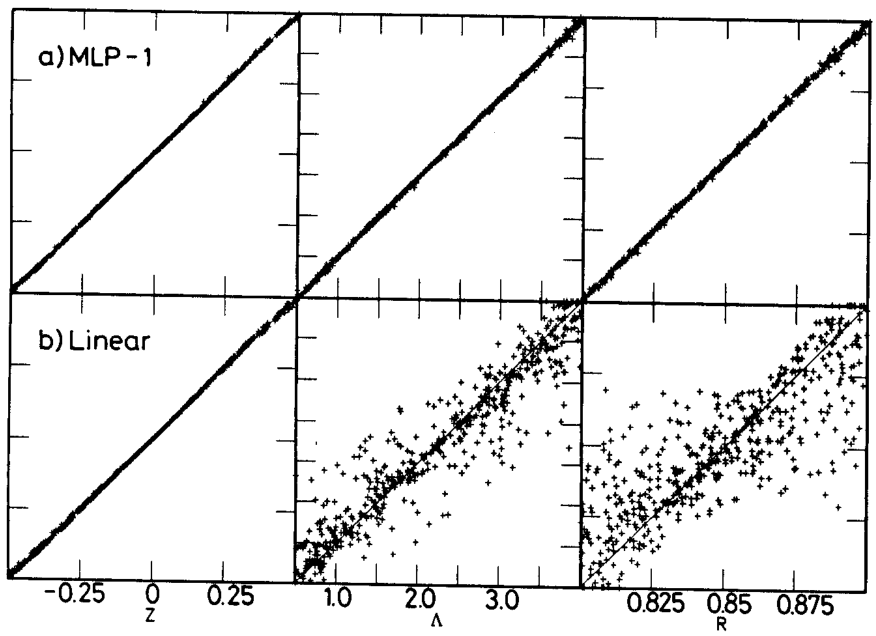


Fig. 10



1
Mean Square Residual
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
+ Linear Mapping
× MLP-1 Mapping
0
10
20
30
40
50
60
70
Number of Flux-loops


Fig. 11



a) MLP-1
b) Linear
-0.25
0
0.25
Z
1.0
2.0
3.0
Λ
0.825
0.85
0.875
R

Fig. 12

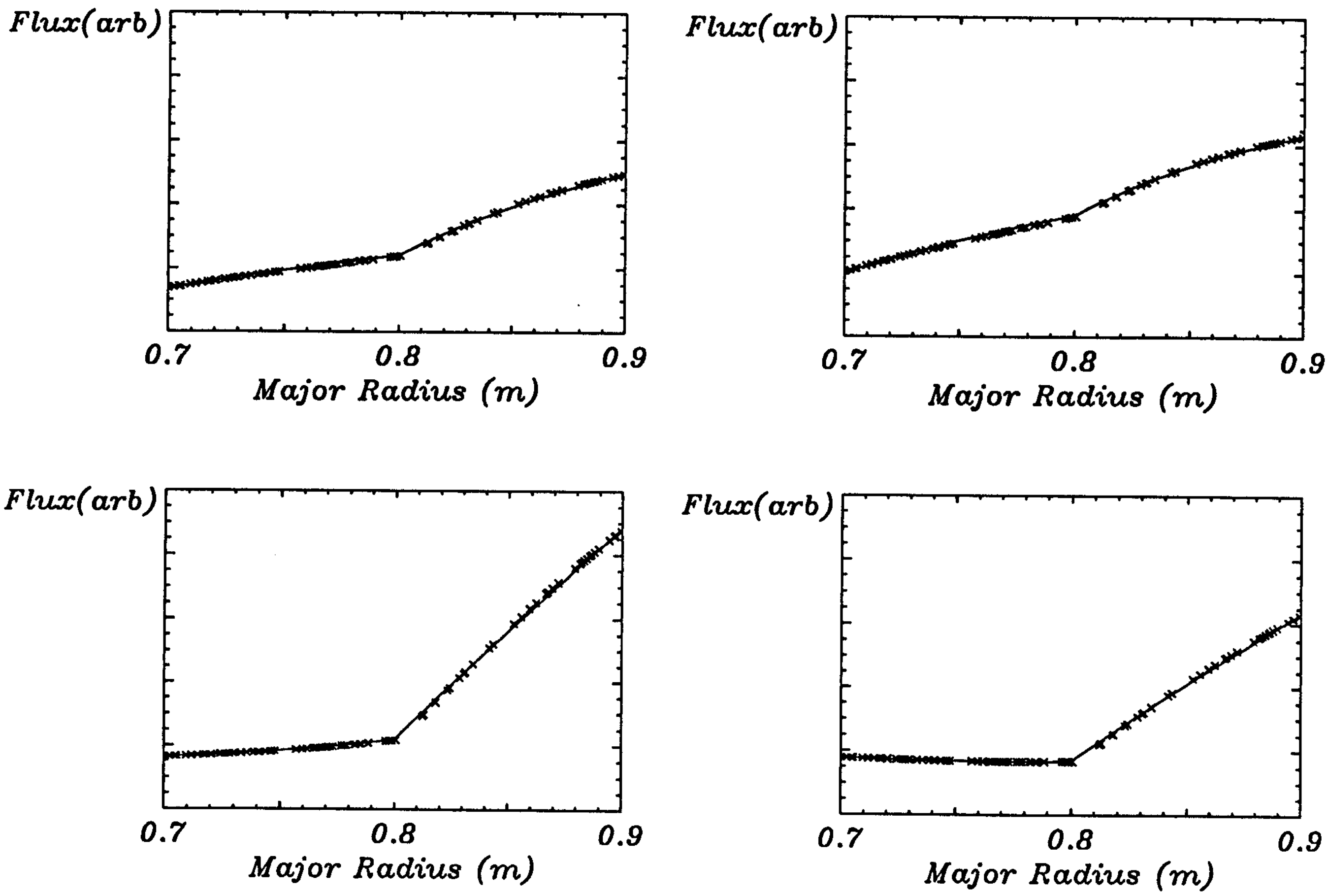
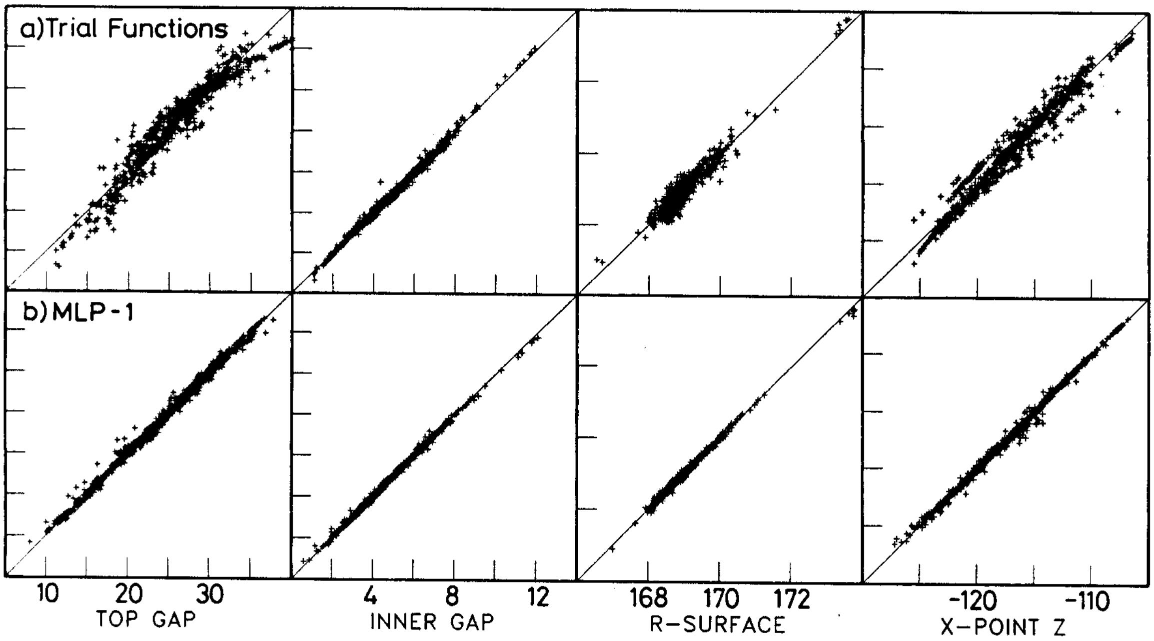
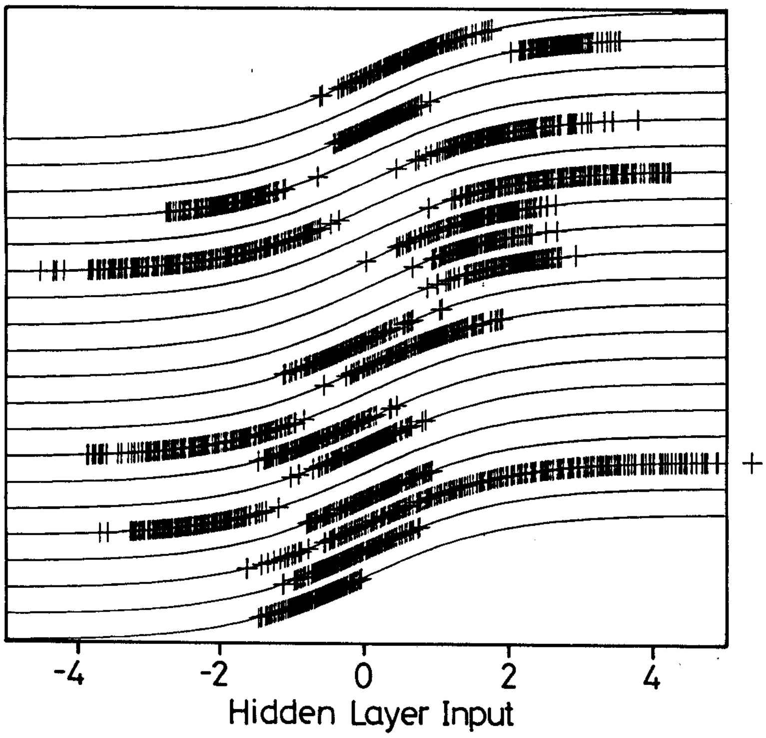

Flux(arb)
0.7
0.8
0.9
Major Radius (m)
Flux(arb)
0.7
0.8
0.9
Major Radius (m)
Flux(arb)
0.7
0.8
0.9
Major Radius (m)
Flux(arb)
0.7
0.8
0.9
Major Radius (m)

Fig. 13



a)Trial Functions
b) MLP-1
10
20
30
TOP GAP
4
8
12
INNER GAP
168
170
172
R-SURFACE
-120
-110
X-POINT Z


Fig. 14



-4
-2
0
2
4
Hidden Layer Input

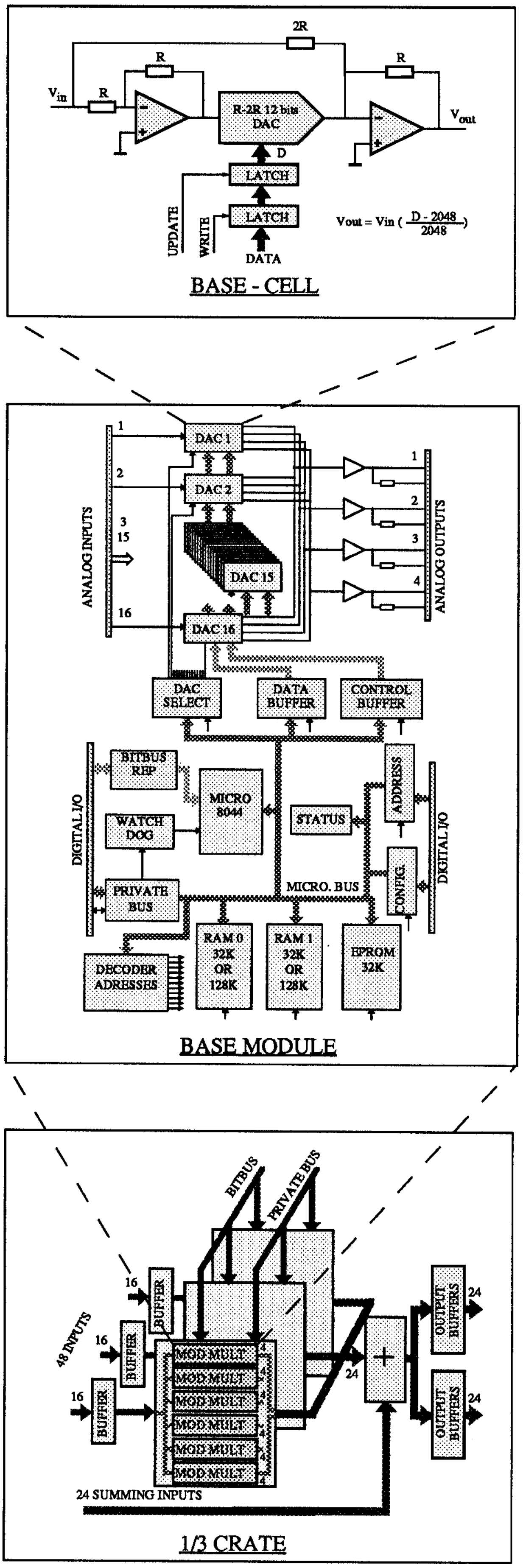

2R
R
R
R
$V_{in}$
R-2R 12 bits DAC
D
$V_{out}$
LATCH
LATCH
UPDATE
WRITE
DATA
Vout = Vin ( D - 2048 / 2048 )
BASE - CELL
ANALOG INPUTS
DAC 1
DAC 2
DAC 15
DAC 16
ANALOG OUTPUTS
DAC SELECT
DATA BUFFER
CONTROL BUFFER
DIGITAL I/O
BITBUS REP
WATCH DOG
MICRO 8044
STATUS
ADDRESS
CONFIG.
MICRO. BUS
PRIVATE BUS
DECODER ADRESSES
RAM 0 32K OR 128K
RAM 1 32K OR 128K
EPROM 32K
BASE MODULE
BITBUS
PRIVATE BUS
48 INPUTS
BUFFER
MOD MULT
OUTPUT BUFFERS
24 SUMMING INPUTS
1/3 CRATE

Fig. 15

Fig. 16

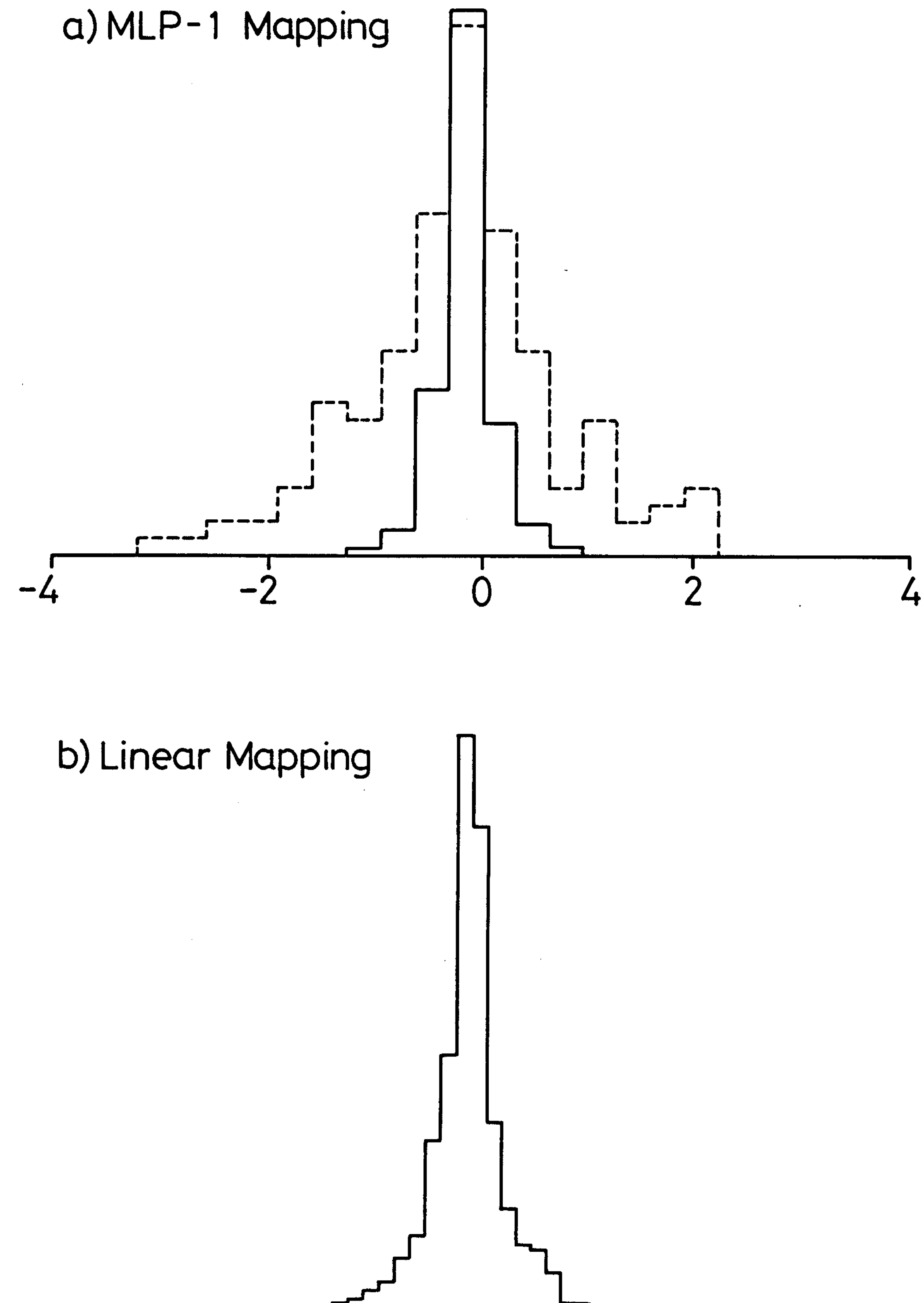


Fig. 17

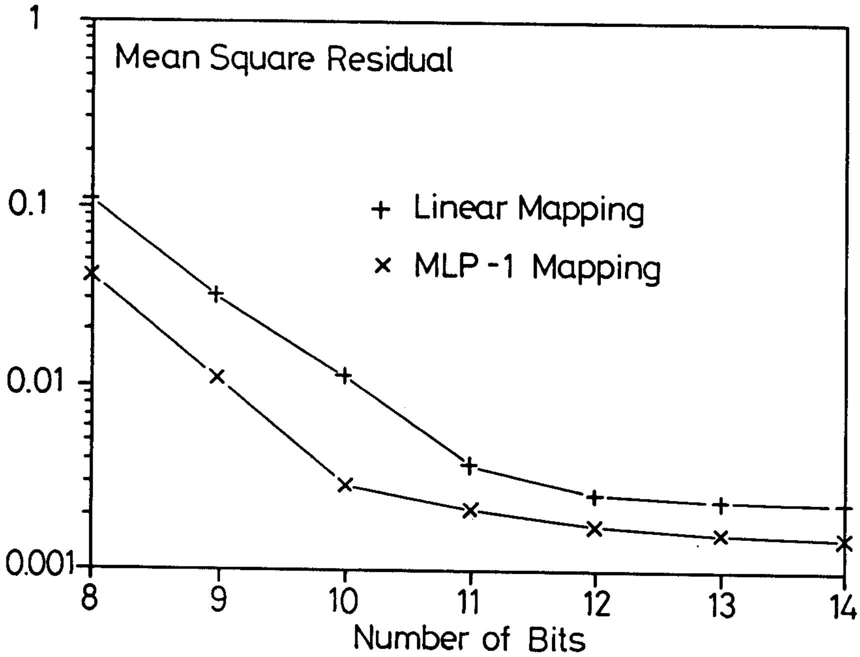

Mean Square Residual
+ Linear Mapping
× MLP -1 Mapping
1
0.1
0.01
0.001
8
9
10
11
12
13
14
Number of Bits

Fig. 18

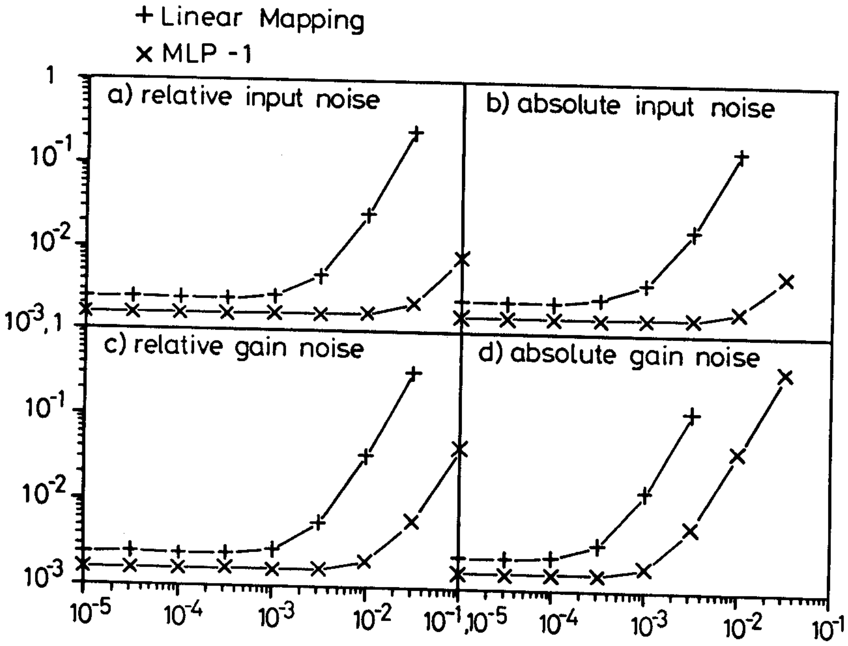

+ Linear Mapping
× MLP -1
a) relative input noise
b) absolute input noise
c) relative gain noise
d) absolute gain noise
1
$10^{-1}$
$10^{-2}$
$10^{-3}$, 1
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-5}$
$10^{-4}$
$10^{-3}$
$10^{-2}$
$10^{-1}$, $10^{-5}$
$10^{-4}$
$10^{-3}$
$10^{-2}$
$10^{-1}$

Fig. 19

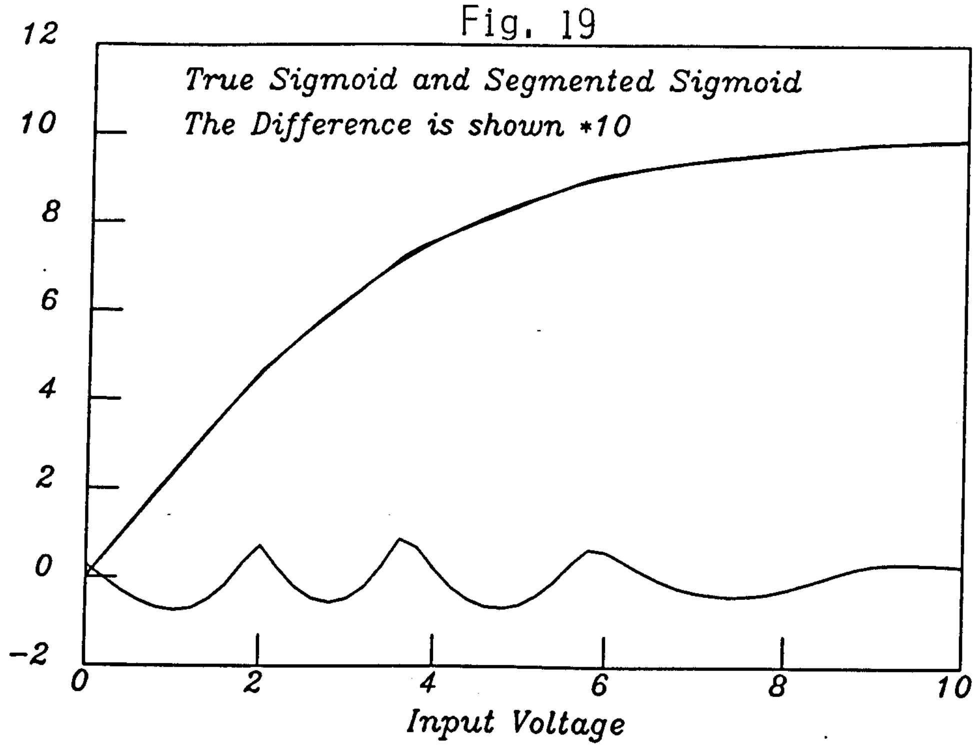
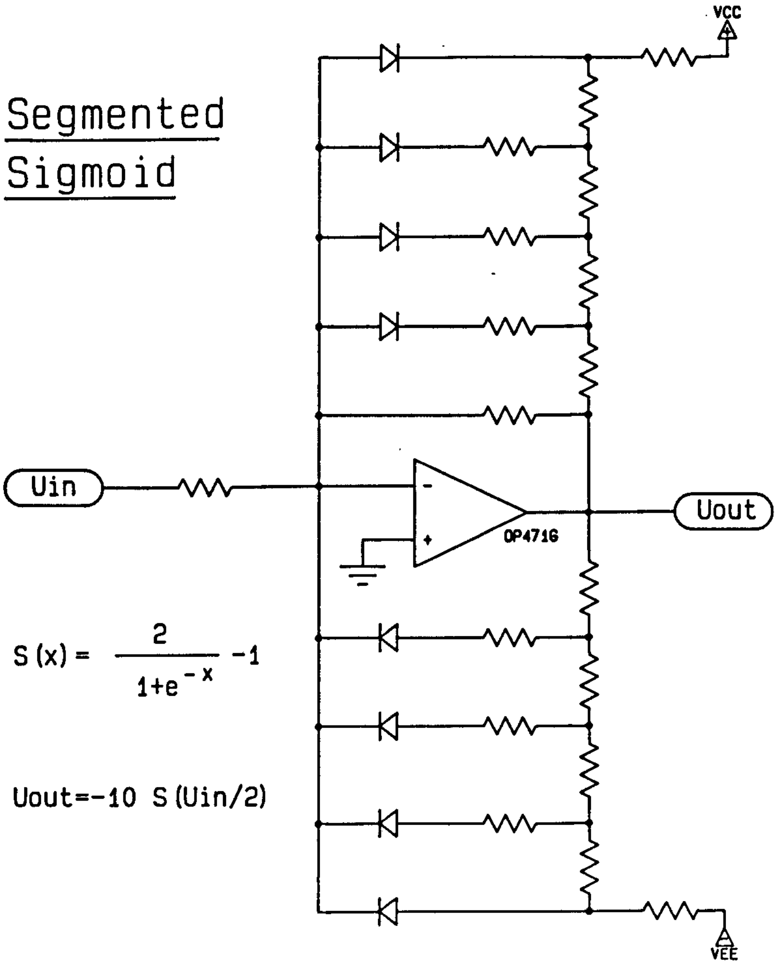



Fig. 20

Segmented Sigmoid




$$S(x) = \frac{2}{1+e^{-x}} - 1$$

Uout=-10 S(Uin/2)