



**Comparing networks with differing neural-node functions
using Transputer based genetic algorithms.**

Antonia J. Jones

Department of Computing, Imperial College London

and

D. Macfarlane

Department of Computer Science, University of Buckingham

DEPARTMENT OF COMPUTING
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY
AND MEDICINE
UNIVERSITY OF LONDON
180 Queen's Gate, London, SW7 2BZ
Telephone: 071-589 5111 Ext: 5017/5096
Telefax: 071-581 8024
E-Mail: ajj@doc.ic.ac.uk

Date/version: 12 October 2002

Published in: Neural Computing & Applications 1(4) 256-267, 1993. ISSN 0-941-0643.

Copyright © 1991. Antonia J. Jones and D. Macfarlane.

Abstract.

Different neural net node computational functions are compared using feedforward backpropagation networks. Three node types are examined: the standard model, ellipsoidal nodes and quadratic nodes. After preliminary experiments on simple small problems, in which quadratic nodes performed very well, networks of differing nodes types are applied to the speech recognition 104 speaker E-task using a fixed architecture. Ellipsoidal nodes were found to work well but not as well as the standard model. Quadratic nodes did not perform well on the larger task. In order to facilitate an architecture independent comparison a transputer based genetic algorithm is then used to compare ellipsoidal and mixed ellipsoidal and standard networks with the standard model. These experiments confirmed the earlier conclusion that ellipsoidal networks could not compare favourably with the standard model on the 104 speaker E-task. In an evolutionary search in which layer node types were free to adjust ellipsoidal nodes had a tendency to become extinct or barely survive. If the presence of ellipsoidal nodes was enforced then the resulting networks again performed poorly when compared with the standard model.

Keywords: Backpropagation, Genetic Algorithms, Ellipsoidal nodes, Quadratic nodes, speech recognition, E-task.

**Comparing networks with differing neural-node functions
using Transputer based genetic algorithms.**

CONTENTS

Introduction.	2
Generalised backpropagation.	3
The output layer calculation.	4
The previous layer calculation.	4
Example 1 (The conventional model).	6
Example 2 (Ellipsoidal model)	7
Example 3 (Quadratic form model)	8
Choice of the output function g	9
The neural net simulation	10
Experiment 1 - A simple test for new node types	10
Experiment 2 - using hidden nodes	11
Experiment 3 - Architecturally fixed comparison on a real world problem	13
Experiment 4 - Adding negative training examples	14
Transputer based Genetic Algorithms	15
Experiment 5 - Evolving mixed node networks	15
Observations and conclusions	17
References	19

List of figures

Figure 1 Feedforward network architecture.	3
Figure 2 The previous layer calculation.	5
Figure 3 Single node ellipsoidal net solution of rotated XOR (an easy problem for an ellipsoidal node).	10
Figure 4 The XOR problem to be solved by 2-2-1 nets of different node types.	11
Figure 5 How the ellipsoidal net solved the XOR problem.	12
Figure 6 Best Guess Error [Test set] v Number of Training patterns.	13
Figure 7 Sum Squared Error [Training set] v Number of Training patterns.	14
Figure 8 As before with counter examples: Best Guess Error [Test Set] v Number of Training patterns.	15
Figure 9 General structure of evolved net.	16
Figure 10 The evolutionary decline of ellipsoidal nodes (typical run).	18

Comparing networks with differing neural-node functions using Transputer based genetic algorithms.

Antonia J. Jones and D. Macfarlane

Introduction.

The primary function of a node is to make a distinction. This is usually done using, in the first instance, a linear function of the inputs to bisect the input space. If the number of inputs is m then this naturally leads to m weights which can be associated with the node or, more conventionally, with the links leading to the node. In this case the activation function is computed as

$$net_j = \sum_{i=1}^m w_{ji} V_i \quad (1)$$

where w_{ij} is the weight from j to i and V_i is the output of node i . The distinction is then made using an output function $g(net_j)$ which can take binary values. If backpropagation is to be used then g must be differentiable and is usually taken to be a sigmoidal function.

In fact, if one is interested in generalising the computational function of a node, it is often convenient to associate the parameters (in the linear case *weights*, let's say) with the node. In which case one just thinks of the links as passing activation values along and one is no longer constrained to have exactly m parameters per node. In [Jones 1992] some simple possibilities for generalisations along these lines were briefly discussed and in this paper we give some specific comparative results. As a simple example of a non-linear net_j , one can have a node which performs its' distinction function by determining whether or not the input vector is within some ellipsoid. In this case there would be m parameters associated with the centre of the ellipsoid and another m parameters associated with the axes. (In addition one could provide the ellipsoid with rotations which would provide further parameters.)

It is arguable that discrimination based on clustering is more appropriate to pattern recognition tasks. Indeed Shepard has constructed a neurophysiological model along these lines [Shepard 1958], [Shepard 1987] and the idea is similar to (but not identical to) radial basis functions which were suggested for similar reasons.

Perhaps the simplest modification to the standard model is to make the slope of the sigmoidal output function an adjustable parameter which can itself be learnt via backpropagation. This adds one parameter per node. It is shown in [Tawel 1989] that a significant decrease in convergence time can be realised by allowing the slope of the sigmoid to actively participate in the learning process in this way.

In this paper we compare the efficiency of network models based on two types of node, linear (i.e. standard model) and ellipsoidal, and briefly discuss the more general model having arbitrary quadratic nodes. For a particular problem, an architecture appropriate for the standard model may be inappropriate for an ellipsoidal model. Thus, in order to compare networks using the two node functions we need to optimise the architecture in each case. We do this using a Transputer based Genetic Algorithm. A survey of Genetic Algorithms applied to neural networks can be found in [Jones 1992]. Here the objects of the GA search are neural networks encoded into gene strings in a suitable fashion and the fitness function of a neural net depends on the accuracy of the trained net and (implicitly) the time required for training. This seems a somewhat novel way to employ a GA which breeds neural networks, but it has enabled us to produce comparative results with a reasonable measure of confidence in the conclusions.

According to Shepard's view we might expect to find that ellipsoidal nodes corresponding to clustering of similar

sensory inputs, perform this second task more efficiently. In order to allow logical predicates of first level sensory data to be formed inhibition is a required capability for neural networks. We observe that inhibition for ellipsoidal nodes arises somewhat differently from the conventional model, but can still occur.

The quadratic node is a natural generalisation of the ellipsoidal node to an arbitrary quadratic form, not necessarily positive definite. This permits a uniform computational model which allows the possibility of ellipsoidal nodes with rotation or hyperbolic sheets which can act effectively as multiple hyperplanes, moreover inhibition can still occur. However, whereas the linear node has m parameters (or $m+1$ if the threshold is included) and the ellipsoidal node has $2m$ parameters, the quadratic node has $\frac{1}{2}m(m+1) + m$ parameters. Clearly networks of such nodes would have to show significant advantages in several directions before the extra complication could be justified.

Ellipsoidal or quadratic nodes allow another interesting issue to emerge: the possibility of using negative training examples in a way that is meaningful in terms of the network model. We had speculated that these networks may be able to exploit negative training examples more efficiently than the standard model. Unfortunately our results indicate that this is not the case and we have not pursued the matter.

Generalised backpropagation.

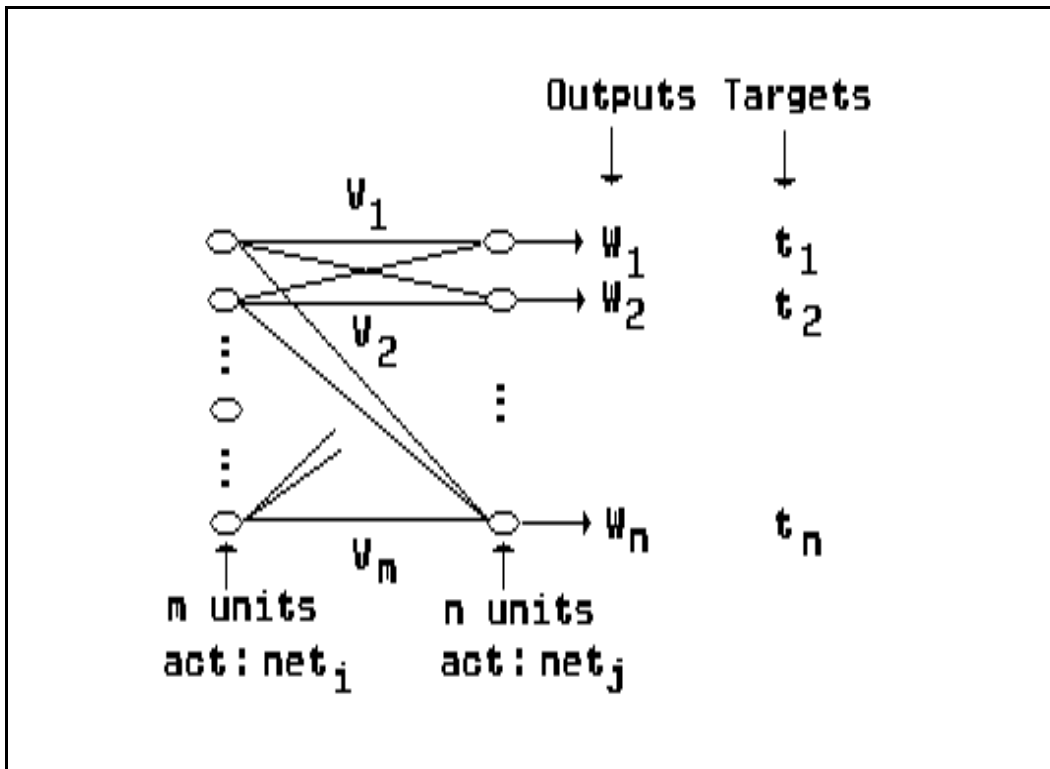


Figure 1 Feedforward network architecture.

The following functions, and hence all their partial derivatives, are assumed known.

Error function.

$$E(W_1, W_2, \dots, W_n, t_1, t_2, \dots, t_n) \tag{2}$$

Here W_1, W_2, \dots, W_n are the outputs from the output layer units and t_1, t_2, \dots, t_n are the target outputs.

Activation function.

Output layer:

$$net_j = net_j (V_1, V_2, \dots, V_m, p_{j1}, \dots, p_{jt}) \quad (3)$$

Here V_1, V_2, \dots, V_m are the outputs from the previous layer and p_{j1}, \dots, p_{jt} are parameters associated with the j th node of the output layer. Frequently $t = t(m)$, i.e. the number of parameters associated with a node is a function of the number of inputs.

Previous layer:

$$net_i = net_i (U_1, U_2, \dots, U_t, p_{i1}, \dots, p_{it}) \quad (4)$$

Here U_1, U_2, \dots, U_t are the outputs from the layer prior to the previous layer.

Output function.

Output layer:

$$W_j = g(net_j) \quad (1 \leq j \leq n) \quad (5)$$

Previous layer:

$$V_i = g(net_i) \quad (1 \leq i \leq m) \quad (6)$$

The output layer calculation.

For the output layer we have

$$\Delta p_{jz} = -\eta \frac{\partial E}{\partial p_{jz}} \quad (7)$$

where $1 \leq z \leq t$, $1 \leq j \leq n$, and η is the learning rate.

Hence

$$\begin{aligned} \Delta p_{jz} &= -\eta \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial p_{jz}} \\ &= \eta \delta_j \frac{\partial net_j}{\partial p_{jz}} \end{aligned} \quad (8)$$

where

$$\begin{aligned} \delta_j &= -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial W_j} \frac{\partial W_j}{\partial net_j} \\ &= -g'(net_j) \frac{\partial E}{\partial W_j} \end{aligned} \quad (9)$$

Equations (8) and (9) express the Δp_{jz} in terms of known quantities.

The previous layer calculation.

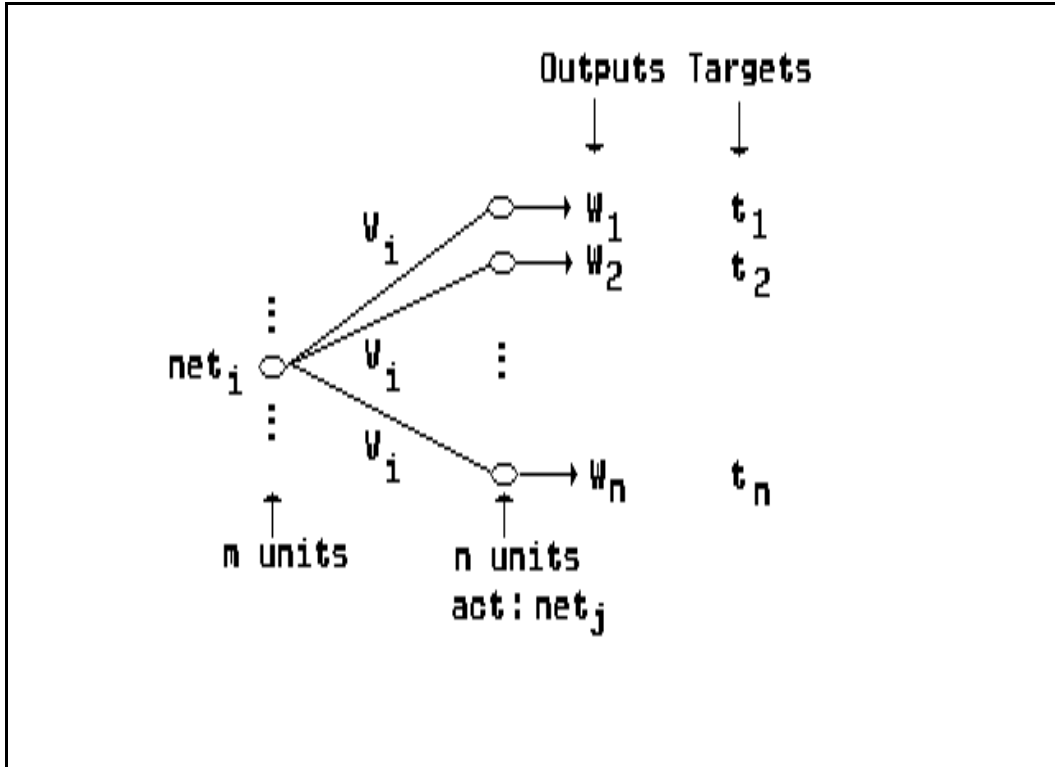


Figure 2 The previous layer calculation.

For the i th node of the previous layer we have similarly

$$\Delta p_{iz} = \eta \delta_i \frac{\partial net_i}{\partial p_{iz}} \quad (10)$$

where

$$\begin{aligned} \delta_i &= -\frac{\partial E}{\partial net_i} = -\frac{\partial E}{\partial V_i} \frac{\partial V_i}{\partial net_i} \\ &= -g'(net_i) \frac{\partial E}{\partial V_i} \end{aligned} \quad (11)$$

However, the partial derivative $\partial E / \partial V_i$ cannot now be evaluated directly. We must evaluate it in terms of other quantities which are known.

Using the chain rule

$$\begin{aligned} \frac{\partial E}{\partial V_i} &= \sum_{j=1}^n \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial V_i} \\ &= -\sum_{j=1}^n \delta_j \frac{\partial net_j}{\partial V_i} \end{aligned} \quad (12)$$

Hence, for node i in this previous layer,

$$\Delta p_{iz} = \eta \delta_i \frac{\partial net_i}{\partial p_{iz}} \quad (13)$$

where

$$\begin{aligned} \delta_i &= -g'(net_i) \frac{\partial E}{\partial V_i} \\ &= g'(net_i) \sum_{j=1}^n \delta_j \frac{\partial net_j}{\partial V_i} \end{aligned} \quad (14)$$

In (13) the partial derivative $\partial net_i / \partial p_{iz}$ is known from (4). In (14) $g'(net_i)$ is known from (6), the δ_j were computed in the previous step, and the last term is known from (3).

Example 1 (The conventional model).

The usual sum squared error is given by

$$E(W_1, \dots, W_n) = \frac{1}{2} \sum_{j=1}^n (W_j - t_j)^2 \quad (15)$$

Hence, for $1 \leq j \leq n$

$$\frac{\partial E}{\partial W_j} = W_j - t_j \quad (16)$$

The linear activation function of (1) becomes

$$\begin{aligned} net_j &= \sum_{i=1}^m w_{ji} V_i & \frac{\partial net_j}{\partial w_{ji}} &= V_i \\ net_i &= \sum_{h=1}^l w_{ih} U_h & \frac{\partial net_i}{\partial w_{ih}} &= U_h \end{aligned} \quad (17)$$

where $t = m$ and $s = l$. Thus (8) becomes

$$\Delta w_{jz} = \eta \delta_j V_z \quad (18)$$

and, using (16), (9) becomes

$$\delta_j = -g'(net_j) \frac{\partial E}{\partial W_j} = -g'(net_j) (W_j - t_j) \quad (19)$$

for an output layer unit.

Similarly (13) becomes

$$\Delta w_{iz} = \eta \delta_i \frac{\partial net_i}{\partial w_{iz}} = \eta \delta_i U_z \quad (20)$$

where, from (14),

$$\begin{aligned}\delta_i &= g'(net_i) \sum_{j=1}^m \delta_j \frac{\partial net_j}{\partial V_i} \\ &= g'(net_i) \sum_{j=1}^m \delta_j w_{ji}\end{aligned}\tag{21}$$

for a hidden layer unit.

Example 2 (Ellipsoidal model).

We use the same error function as before *viz.* (15) and (16). However, the activation function is now

$$\begin{aligned}net_j &= \sum_{i=1}^m A_{ji}^2 (V_i - c_{ji})^2 \\ \frac{\partial net_j}{\partial V_i} &= 2A_{ji}^2 (V_i - c_{ji})\end{aligned}\tag{22}$$

where the A_{ji} terms are squared to ensure that the quadratic form is positive definite, thus $net_j \leq 1$ defines a closed ellipsoidal region of the input space.

Also

$$\begin{aligned}\frac{\partial net_j}{\partial c_{ji}} &= -2A_{ji}^2 (V_i - c_{ji}) \\ \frac{\partial net_j}{\partial A_{ji}} &= 2A_{ji} (V_i - c_{ji})^2\end{aligned}\tag{23}$$

So taking $t = 2m$ and

$$(p_{j1}, \dots, p_{jm}) = (c_{j1}, \dots, c_{jm})$$

$$(p_{jm+1}, \dots, p_{j2m}) = (A_{j1}, \dots, A_{jm})$$

equation (8) becomes

$$\begin{aligned}\Delta c_{jz} &= -\eta \delta_j 2A_{jz}^2 (V_z - c_{jz}) & (1 \leq j \leq n, 1 \leq z \leq m) \\ \Delta A_{jz} &= \eta \delta_j 2A_{jz} (V_z - c_{jz})^2 & (1 \leq j \leq n, m+1 \leq z \leq 2m)\end{aligned}\tag{24}$$

where, from (9),

$$\delta_j = -\frac{\partial E}{\partial net_j} = -(W_j - t_j) g'(net_j) \quad (1 \leq j \leq n)\tag{25}$$

for an output layer unit.

Similarly, (13) becomes

$$\begin{aligned}\Delta c_{iz} &= \eta \delta_i \frac{\partial net_i}{\partial c_{iz}} = -\eta \delta_i 2A_{iz}^2 (U_z - c_{iz}) \\ \Delta A_{iz} &= \eta \delta_i \frac{\partial net_i}{\partial A_{iz}} = \eta \delta_i 2A_{iz} (U_z - c_{iz})^2\end{aligned}\tag{26}$$

where, from (14),

$$\begin{aligned}\delta_i &= g'(net_i) \sum_{j=1}^n \delta_j \frac{\partial net_j}{\partial V_i} \\ &= g'(net_i) \sum_{j=1}^n \delta_j 2A_{ji}^2 (V_i - c_{ji})\end{aligned}\tag{27}$$

Of course there is no particular reason why the learning rate η need be the same for both types of parameter.

Note that (22) permits a previous node to *inhibit* (i.e. reduce) the output of a node in the next layer. Although a single input cannot make a negative contribution to the activation function it can nevertheless cause the vector \mathbf{V} of inputs to lie outside the region in which the output function produces a response near one, thus inhibiting the node. Moreover, inhibition of other nodes is still possible even if the associated form is not positive definite (which would seem quite likely, since the probability that a diagonalised quadratic form of dimension m has all coefficients positive is $(\frac{1}{2})^m$). Consequently it is possible for such networks to form more complicated representations using the hidden units just as in the conventional model.

Example 3 (Quadratic form model).

We use the same error function as before, viz. (15) and (16). However, the activation is now

$$\begin{aligned}net_j &= \frac{1}{2}(\mathbf{V}-\mathbf{c}_j) \mathbf{A}_j (\mathbf{V}-\mathbf{c}_j)^T = \frac{1}{2} \sum_{u,v} (V_u - c_{ju}) A_j^{uv} (V_v - c_{jv}) \\ \frac{\partial net_j}{\partial V_i} &= \frac{1}{2} \left(\sum_{u \neq i} (V_u - c_{ju}) A_j^{ui} + 2A_j^{ii} (V_i - c_{ji}) + \sum_{v \neq i} A_j^{iv} (V_v - c_{jv}) \right) = \sum_w A_j^{wi} (V_w - c_{jw})\end{aligned}\tag{28}$$

where \mathbf{V} is the m -dimensional vector of inputs, $\mathbf{A} = (A^{uv})$, $1 \leq u, v \leq m$, is a symmetric parameter matrix, not necessarily positive definite, and superscript T denotes transpose. Also

$$\begin{aligned}\frac{\partial net_j}{\partial c_{jz}} &= - \sum_w A_j^{zw} (V_w - c_{jw}) \\ \frac{\partial net_j}{\partial A_j^{uv}} &= (V_u - c_{ju})(V_v - c_{jv})\end{aligned}\tag{29}$$

where we have again used the symmetry of the matrix.

Now $t = m + \frac{1}{2} m(m+1)$, $p_{jz} = c_{jz}$ for $1 \leq z \leq m$ and $p_j^{uv} = A_j^{uv}$, so that (8) for output units becomes

$$\begin{aligned}\Delta c_{jz} &= -\eta \delta_j \sum_w A_j^{zw} (V_w - c_{jw}) \\ \Delta A_j^{uv} &= \eta \delta_j (V_u - c_{ju})(V_v - c_{jv})\end{aligned}\tag{30}$$

where from (9) and (15), as before,

$$\delta_j = -\frac{\partial E}{\partial net_j} = -(W_j - t_j) g'(net_j) \quad (1 \leq j \leq n) \quad (31)$$

Similarly, (13) becomes

$$\begin{aligned} \Delta c_{iz} &= -\eta \delta_i \sum_w A_i^{zw} (U_w - c_{iw}) \\ \Delta A_j^{uv} &= \eta \delta_i (U_u - c_{iu})(U_v - c_{iv}) \end{aligned} \quad (32)$$

where from (14), using (28)

$$\delta_i = g'(net_i) \sum_{j=1}^n \delta_j \sum_w A_j^{wi} (V_w - c_{jw}) \quad (33)$$

Choice of the output function g.

For the first example of a linear activation function any sigmoidal function g is suitable. Frequently the function

$$W_j = g(net_j) = \frac{1}{1 + e^{-net_j + \theta_j}} \quad (34)$$

is used. Here θ_j is the threshold, or bias, of the unit. Conventionally the threshold is treated as just another weight by creating a dummy unit which is always on (somewhat like ground in an electrical circuit). However, from the present viewpoint the threshold can be considered as just another parameter associated with the unit, in which case there is no need for the dummy unit.

For the ellipsoidal model we require a function g which is large (i.e. close to 1) when net_j is small, and close to zero when net_j is large. A suitable function might be for example

$$W_j = g(net_j) = e^{-net_j} \quad (35)$$

We might expect to square the exponent, however this is not really necessary since we have already taken steps to ensure that $net_j \geq 0$.

For the quadratic model, for similar reasons, we take

$$W_j = g(net_j) = e^{-net_j^2} \quad (36)$$

The neural net simulation.

An ANN simulator has been developed that can perform backpropagation on an arbitrary feedforward network with mixed node types. Any node can be specified to be standard, ellipsoidal or quadratic. The software is structured so that adding new node types is straightforward. The essential elements defining a node type are:

- The activation function and its derivative.
- A procedure for propagating its contribution to the $\partial net / \partial V$ terms of nodes that supply inputs (e.g. equation (27) for ellipsoidal nodes).

- The rule to calculate Δ -parameters (e.g. equation (32) for quadratic nodes).

The parameter update procedure follows the common practice of including a momentum term alpha so that (10) becomes

$$\Delta p_{iz}(t+1) = \eta \delta_i \frac{\partial net_i}{\partial p_{iz}} + \alpha \Delta p_{iz}(t) \quad (37)$$

Experiment 1 - A simple test for new node types.

The first step was to check that the ellipsoidal and quadratic node types could solve a small problem to which they are well suited. The problem chosen has two classes with the same geometric relationship as the XOR problem except rotated by 45 degrees. It is shown in Figure 3.

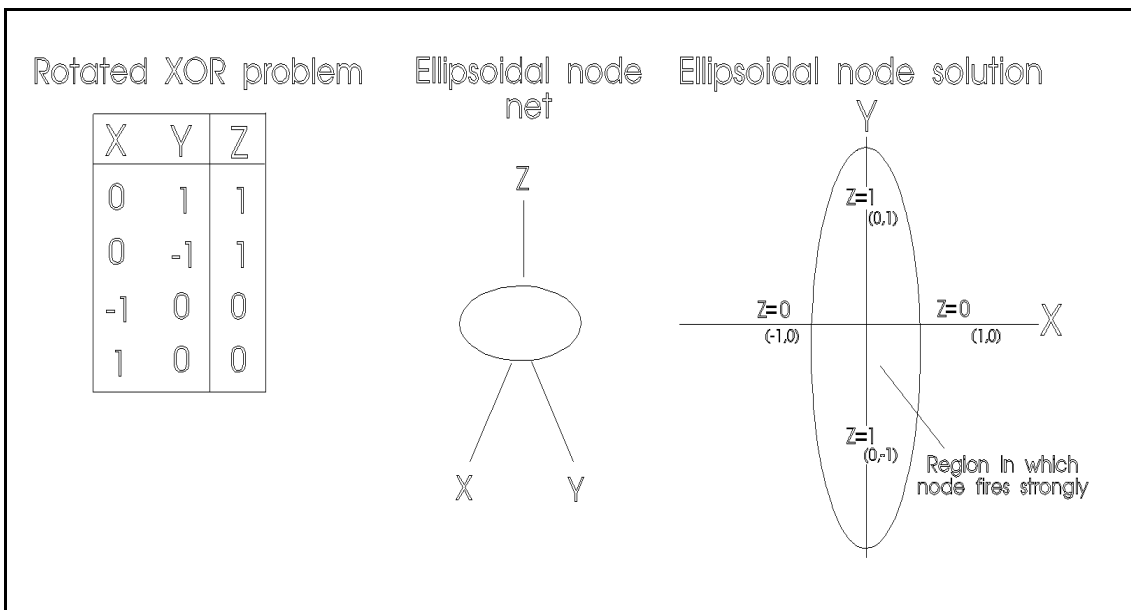


Figure 3 Single node ellipsoidal net solution of rotated XOR (an easy problem for an ellipsoidal node).

Only a single ellipsoidal node should be needed to define an ellipse that separates the two classes, as shown in Figure 3. This was indeed the case, the single ellipsoidal reduced the sum squared error to below 0.1 within several training epochs when using a learning rate of 1.0. No momentum was used.

As expected the problem was also quickly solved by a single quadratic node, using the same parameters as the ellipsoidal one. This is due to the quadratic node using cross products between the two inputs when calculating the net input.

Experiment 2 - using hidden nodes.

The first problem had no need of hidden nodes. The simplest problem for which both standard and ellipsoidal nodes need hidden units is the standard XOR problem, shown in Figure 4. The reason that a single ellipsoidal node cannot solve it is because ellipsoidal nodes have no rotation parameters for the ellipses they define. Of course, a single quadratic node can easily solve the standard XOR problem. However, it seemed worthwhile to compare all three nodes types with the same architecture.

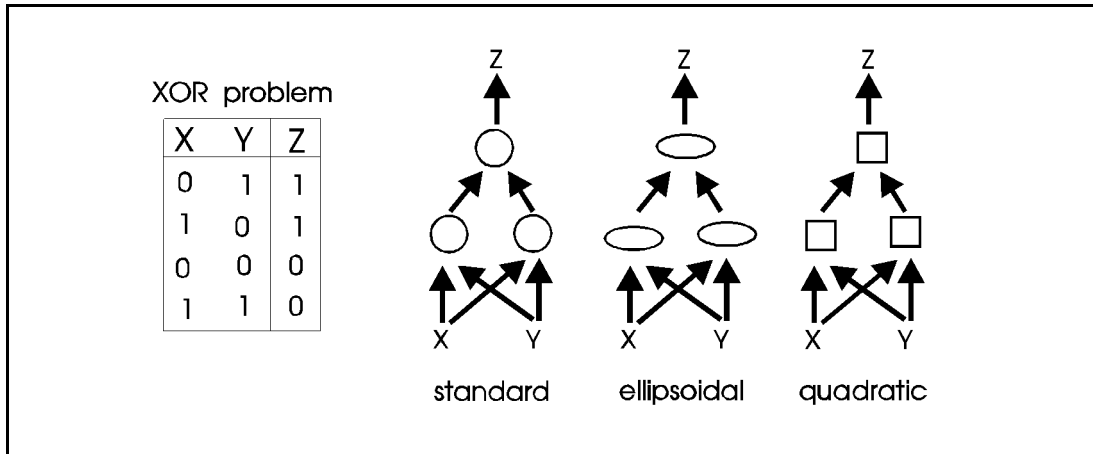


Figure 4 The XOR problem to be solved by 2-2-1 nets of different node types.

Networks consisting of two hidden nodes and an output node (2-2-1) were constructed for each node type; standard, ellipsoidal and quadratic. These were then tested on the XOR problem. All networks were capable of solving the problem.

Each network was run for up to 50,000 cycles through the training set, updating the weights after each cycle. Three learning rate values were tried: $\eta = 0.9, 0.1, 0.01$. For each setting 20 runs were made using different random weight starts. Weights were initialised to be between -0.5 and 0.5. For each run it was noted if the sum squared error was reduced to below 0.1 within the 50,000 cycles and if so how many cycles were needed. It was noted that a high momentum term benefited all of the networks for this problem, so for this experiment a momentum of $\alpha = 0.9$ was used.

Table 1. 2-2-1 Networks on the XOR problem.

Node	Standard			Ellipsoidal			Quadratic		
η	0.9	0.1	0.01	0.9	0.1	0.01	0.9	0.1	0.01
%cvg	95	100	100	0	100	90	65	100	100
Max	370	2286	22539	-	3817	28048	91	30	166
Min	102	667	6301	-	53	318	6	17	75
Avg	197	1215	11755	-	506	3314	17	23	117

Key

%cvg is the percentage of runs reaching < 0.1 sum squared error.

Max, Min and Avg are the maximum, minimum and average number of cycles taken to reduce the sum squared error below 0.1.

As can be seen the standard node net solves the problem consistently and prefers a high learning rate. The time to solution increases steadily as the learning rate is decreased.

The ellipsoidal net converges prematurely using a learning rate of 0.9 but works well at lower learning rates, outperforming the standard by over a factor of two. It is worth noting however that the ellipsoidal nets would sometimes jump to a high sum squared error score after reducing it below 0.1.

The best performance as expected for this problem is from the quadratic nodes. The quadratic net was prone to getting stuck in poor local minima for the highest learning rate value but consistently solved the problem in tens of epochs at lower learning rates.

It is interesting to examine the solution produced by the ellipsoidal net. Intuitively it would seem that the output unit should perform some sort of union operation on regions defined by the hidden units, firing if the input fell within any of the hidden unit regions. In order to do this a logical OR of the input units is required. This is not easy to achieve using an ellipsoidal node because it must define an ellipse that encompasses all corners on the hypercube defined by its inputs, except the corner where all inputs are off.

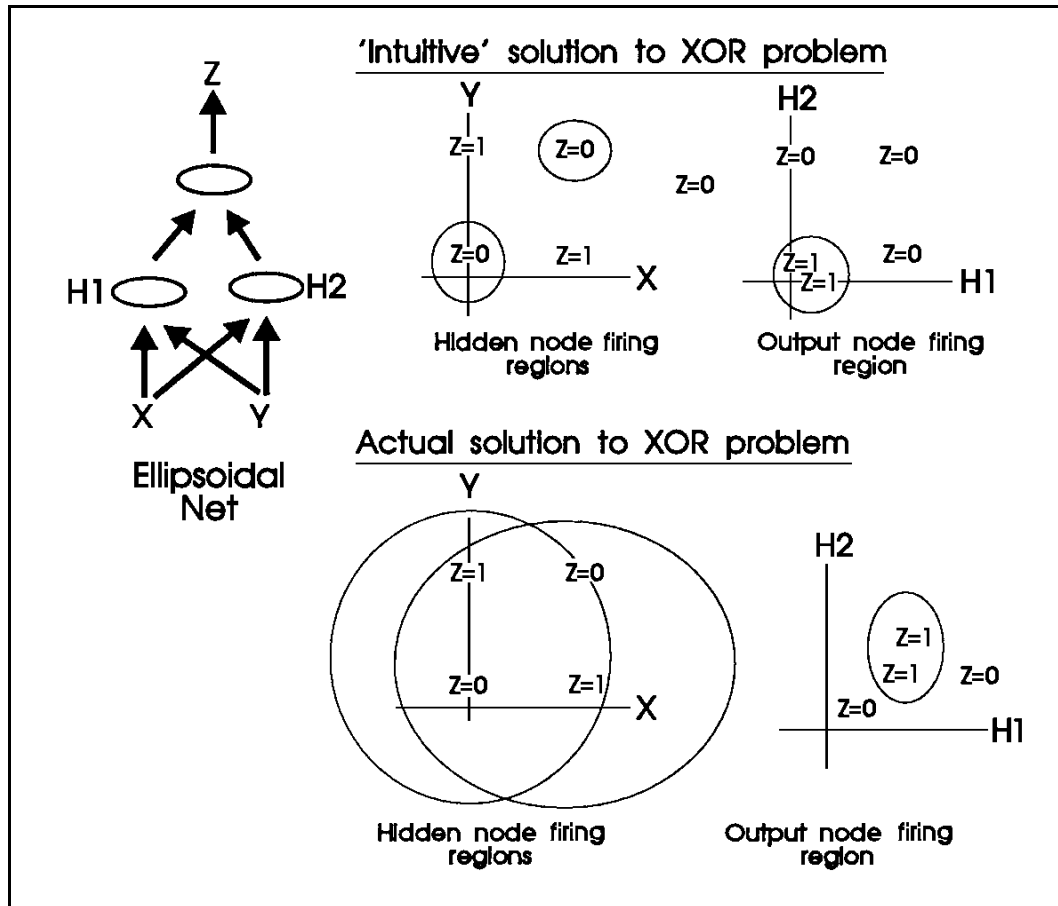


Figure 5 How the ellipsoidal net solved the XOR problem.

The corners where at least one input has fired will also vary widely with respect to their distances from the centre of the ellipse. A logical AND is much easier, corresponding to a single corner of the input hypercube (all on). By this reasoning a perfect solution for the XOR problem was hand-crafted and is shown in Figure 5 (upper). The actual solution produced by backpropagation turned out to be quite different.

The backpropagation solution produced hidden node regions that overlapped and encompassed the space of all training examples. The distances of the training examples from the centres of the hidden nodes ellipses produced a mapping from the inputs to the hidden unit outputs that was a bit like a crude rotation. This rotated space allowed the classes to be easily separated by the output unit, much as in the original easy problem experiment. This is pictured in Figure 5 (lower).

Experiment 3 - Architecturally fixed comparison on a real world problem.

The performance of the three node types were then tested on a "real world" problem. The data used was extracted from the British Telecom Connex database and consists of eight E-sounds: B,C,D,E,G,P,T, and V produced by each of 104 speakers. The sounds are represented by 26 frames each of 8 cepstral coefficients. This data has been the object of a large comparative study of speech recognition techniques [Bimbot 1990].

Although a difficult problem to solve to very high accuracy of classification using ANN techniques it has been previously discovered that a standard node network with no hidden units (208-8) can produce quite reasonable

scores. To perform a comparison three 208-8 networks, one for each type of node were constructed. The standard net had $(208 * 8) + 8 = 1672$ parameters, the ellipsoidal net had $2 * 208 * 8 = 3328$ parameters while the quadratic net had $8 * (\frac{1}{2} * 208 * 209 + 208) = 175,552$ parameters.

In testing the nets, the E-sound data was divided into two halves, one used for training and the other for generalisation testing. Each network was trained for 100,000 training vector presentations. For all networks α was set to 0.9, for the standard model a learning rate of 0.1 was used, while in the other two it was set to 0.01.

At regular intervals the network was tested to reveal its sum squared error and also best guess error scores over both the training and test sets. The best guess error is of more interest here than the sum squared error score, indicating the proportion of examples misclassified through producing outputs closer (in Euclidean distance) to targets other than the correct target.

The quadratic nodes performed disappointingly by very rapidly becoming stuck at high sum squared error scores and no better than random classification. Attempts were made improve the situation by first initialising weights to very small values (to restrict initial net inputs) and then trying a sparsely connected 208-102-34-8 net where each hidden node was connected to only four in the previous layer (to reduce the large number of parameters). These changes made little difference to the poor performance.

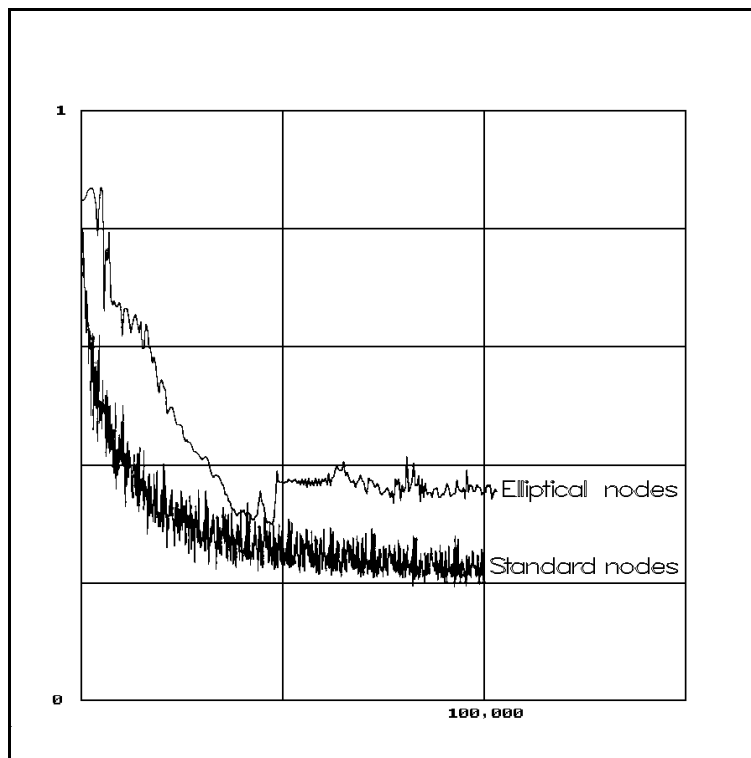


Figure 6 Best Guess Error [Test set] v Number of Training patterns.

Shows Standard and Ellipsoidal 208-8 networks.

Standard net: $\eta = 0.1, \alpha = 0.9$.

Ellipsoidal net: $\eta = 0.01, \alpha = 0.9$.

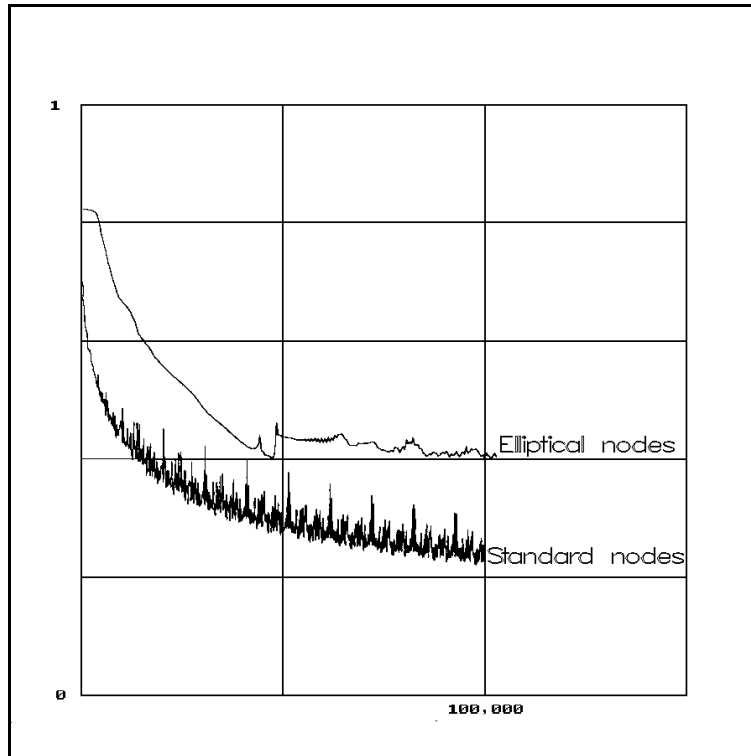


Figure 7 Sum Squared Error [Training set] v Number of Training patterns.

Shows Standard and Ellipsoidal 208-8 networks.

Standard net: $\eta = 0.1, \alpha = 0.9$.

Ellipsoidal net: $\eta = 0.01, \alpha = 0.9$.

In the small number of runs the standard and ellipsoidal fared much better. Their performances are compared on the best guess error (test set) in Figure 6 and sum squared error (training set) in Figure 7. The sum squared error of the standard net fluctuates cyclically with a downward trend. The overall shape of the error curve is inverse exponential. The sum squared error curve for the ellipsoidal net has fewer fluctuations but levels off at a higher error score. The generalisation to classification of the test set can be seem to follow a similar patterns. The best score produced by a standard net gave just over 80% [Test set] correct classification while the best ellipsoidal score was about 69% [Test].

The result of these tests show the quadratic nodes failing to cope with the problem while the ellipsoidal nodes perform respectably, a little worse than the standard nodes.

Experiment 4 - Adding negative training examples.

Historically it has often been remarked (in the context of a feedforward conventional model applied to pattern recognition for some given number of classes) that negative training examples cannot easily be used in backpropagation networks. Upon further examination we see that this objection really arises from the following fact. Consider, for example, a single layer network and suppose we assign some vector of outputs (the vector $(0,0,\dots,0)$ comes to mind) to signify that a given training input does not fall into any of the classes we seek. For hyperplane nodes *a large number of such training exemplars could be fatal*, since there could soon be no orientation of the hyperplanes which would give consistent classification, even if the original data samples were linearly separable. However, in principle, the ellipsoidal or quadratic node models can respond to a large class of negative training exemplars by adapting the coefficients of the quadratic forms in the input layer so that they are positive definite and exclude the counter examples where necessary. Indeed, according to this arguement a large number of counter examples should *improve* the performance of such networks.

We decided to test the hypothesis that ellipsoidal nodes may outperform standard nodes in the case that there are

many counter examples by modifying the E-sound data to contain counter examples (randomly generated speech data can safely be assumed to be counter examples). Random input patterns were generated and assigned a new class, output (0 0 0 0 0 0 0). One hundred of these were added to the training set and the same number to the test set. The previous experiments were then repeated with the new data.

The best guess error (test set) plot is shown in Figure 8. The random input patterns have decreased the performance of *both* the standard and ellipsoidal nets. The best ellipsoidal net classification score is around 62% and for the standard about 70%. Both differences between the scores are similar to those before adding counter examples, so no evidence of the hypothesis can be seen here. Further investigation is evidently needed.

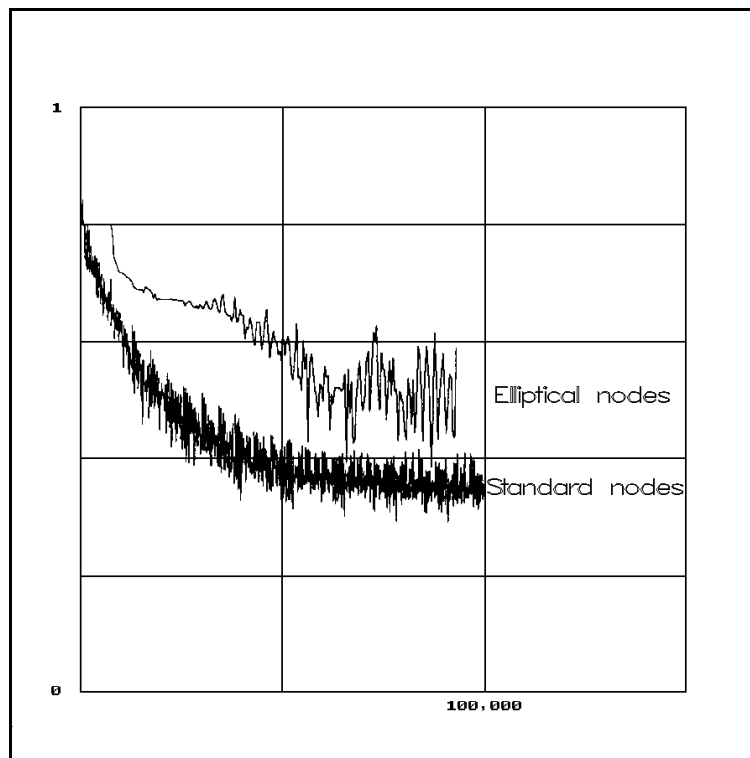


Figure 8 As before with counter examples: Best Guess Error [Test Set] v Number of Training patterns.

Note: For this experiment 100 random input patterns were paired with a 'junk' class (0,0,0,0,0,0,0) output and added to the training set. Another 100 were added to the test set.

Shows Standard and Ellipsoidal 208-8 networks.

Standard net: $\eta = 0.1, \alpha = 0.9$.

Ellipsoidal net: $\eta = 0.01, \alpha = 0.9$.

Comment: Unfortunately ellipsoidal nodes seem to degrade in performance to the same extent as standard nodes (compare with the previous graph).

Transputer based Genetic Algorithms.

Experiment 5 - Evolving mixed node networks.

In order to more fully explore the suitability of ellipsoidal nodes and perhaps gain some insight into the types of architecture they would be suited to, a network structure optimisation system was used to search through configurations of mixed node networks for solving the E-sound problem. The network structure optimisation system consists of a parallel genetic algorithm executed on network of transputers (an MIMD parallel processing system). The justification for this approach to network structure optimisation and a description of the network types can be

found in [Dodd 1991]. Parallel genetic algorithms are discussed in [Macfarlane 1989].

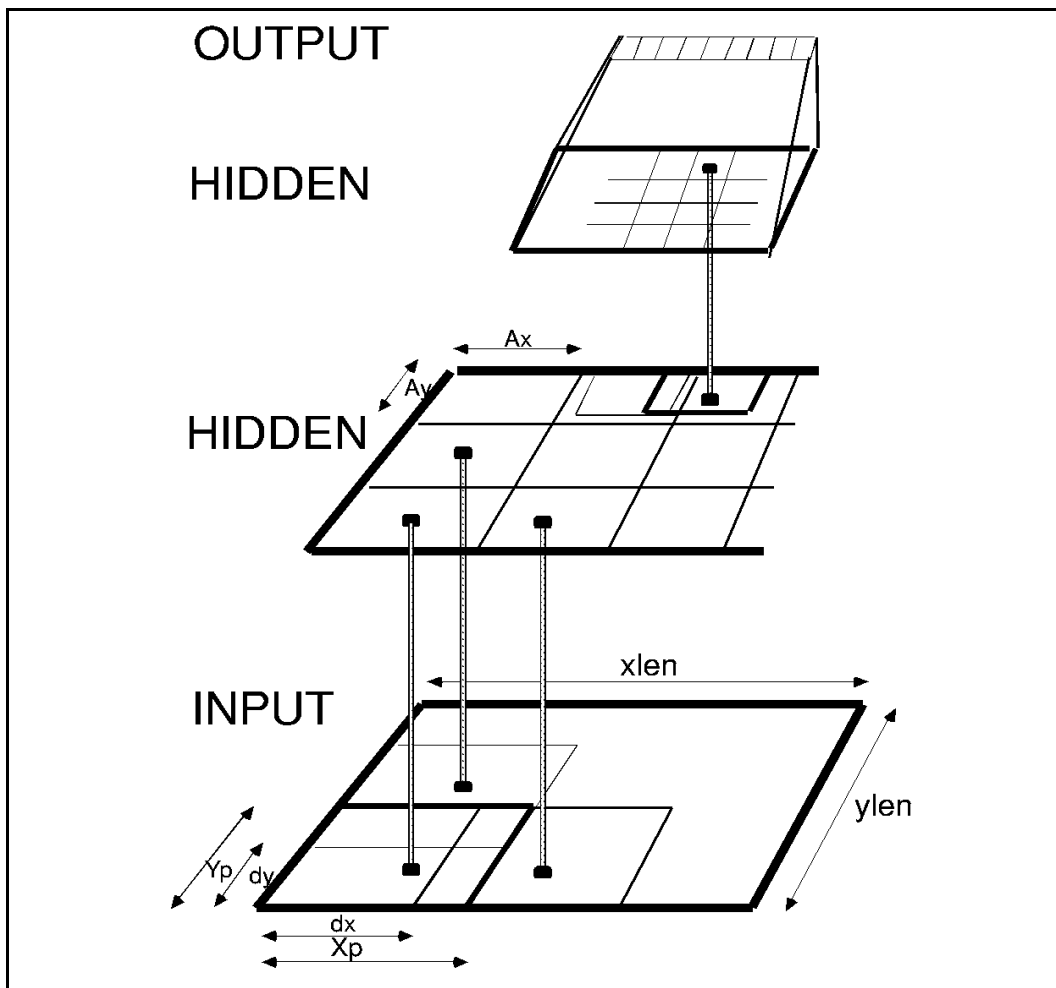


Figure 9 General structure of evolved net.

The space of candidate networks is parameterised in a flexible fashion, covering the space of many numerous network structures such as fully connected layered nets, scaly nets and TDNN nets. The general network structure is shown in Figure 9. It consists of several layers each consisting of units arranged in rectangles. Layers are tiled by patches of units (possibly overlapping) which are fully connected to patches in the layer above. The last hidden layer is fully connected to the output units. Thus there is a huge degree of flexibility in the fine grained architecture possible.

A key aspect in genetic optimisation is the encoding used. In this case the encoding is a parameterisation of network features for each layer, represented as a binary string. These parameters include the patch sizes and overlap in the x and y dimensions, the type of nodes (standard or ellipsoidal) in a given layer, and the activation function type (logistic, gaussian and sinusoidal functions possible for standard nodes). There are also expression flags which allow whole layers to be switched on and off and a few global parameters affecting training (the learning rate and a momentum term). Another feature is that weights associated with links terminating at any patch can optionally be shared for all of the patches in a layer thus reducing the number of distinct weights to modify. The sharing is also switched by a flag.

The genetic algorithm used a structured population of sixteen networks. Each network was trained for 30 minutes only and 30 generations were performed. The evaluation criterion for a network was based on the best guess scores.

$$\text{Fitness} = 1/(1+\text{best-guess-training}) + 2/(1+\text{best-guess-test})$$

Our main experiment allowed the networks produced to have up to three hidden layers each layer consisted of either

entirely ellipsoidal or entirely standard nodes. The characteristics of the best networks of each type discovered were noted in the following table.

Table 2. Best networks found by genetic search

Best network	1	2
Number of hidden layers	1	1
Total number of nodes	304	312
Number of hidden nodes	88	96
Number of ellipsoidal nodes	0	96
Number of links	3168	2496
Number of weights/parameters	1031	920
Best %correct for training data	90%	82%
Best %correct for test data	82%	76%

Key

1 = Standard: Logistic $\eta = 0.39$, $\alpha = 0.62$

2 = Mixed: Ellipsoidal hidden $\eta_A = 0.01$, $\alpha_A = 0.11$, $\eta_c = 0.29$, $\alpha_c = 0.61$ (also applied to logistic nodes).

Note: Both networks evolved to use weight sharing so the number of links and parameters is stated separately.

In test one *purely ellipsoidal networks became extinct and only one mixed node network survived to the end of the run*, see Figure 10. The best network produced had one hidden layer that used shared weights. All nodes were standard logistic and learning rate was quite large. Performance of the network is as good as can be expected for the speech data. The reason that ellipsoidal nodes could not compete effectively may be due to their increased computational overhead and the fact that a fixed training time was imposed. The mixed node network that survived implemented weight sharing, and this was no doubt an important factor in its fitness which at best is marginal. In other evolutionary runs ellipsoidal nodes became extinct, not surviving long enough to evolve a useful form of weight sharing. In other experiments in which the presence of an ellipsoidal layer was *enforced* the performance of such nets remained poor.

Quadratic nodes were not considered due to their storage demands in relation to the limited memory available for

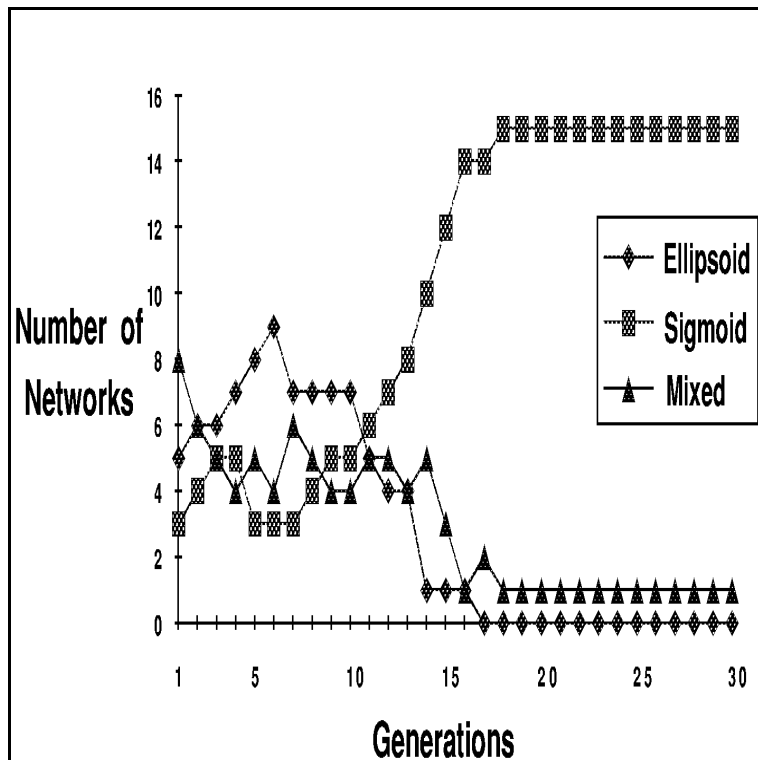


Figure 10 The evolutionary decline of ellipsoidal nodes (typical run).

each processor, and because they performed so poorly in the preliminary tests.

Observations and conclusions.

A single quadratic node can solve the XOR problem very easily. For *small* 2-2-1 networks solving the XOR problem the quadratic node network converged *extremely* rapidly compared with the standard model. The ellipsoidal net with high learning rate failed, but at low learning rates also converged more rapidly than the standard model.

In a more realistic test using 208-8 networks on the 104 speaker E-problem these results were *not* maintained. The quadratic net failed utterly, even when modified to a 208-102-34-8 sparsely connected network with far fewer parameters. The standard and ellipsoidal nets did much better but the ellipsoidal net (with far more parameters) had a higher error score than the standard model.

Comparing standard and ellipsoidal networks using the transputer based genetic algorithm confirmed the earlier conclusion that ellipsoidal networks could not compare favourably with the standard model on the 104 speaker E-task. In an evolutionary search in which layer node types were free to adjust, networks of purely ellipsoidal nodes rapidly became extinct, although one mixed node network did survive. The speech task is an interesting one because in practice and in theory a good solution involves sub-nets of the network sharing weights. It is notable that the only mixed node net to survive did exploit this feature, but even so did not perform as well as the best standard network. In other evolutionary runs mixed node and purely ellipsoidal networks did not survive long enough to develop useful weight sharing and so became extinct.

It seems safe to conclude that the burden of the extra parameters required by the ellipsoidal and quadratic models in large networks excessively increases the complexity of the search space resulting in less numerical stability, longer learning times and in generally poorer performance. Whilst the results of [Tawel 1989] show that some improvement on the standard model by adding learnable parameters is possible, our results suggest that any modification of the standard model should be as simple as possible and involve the addition of only a small number of parameters (in Tawel's case only one parameter was added).

In general 'higher order nets' usually means that net_i has some non-linear terms, typically $w_{ijk}V_jV_k$, and such nets are interesting and useful because a second order or higher net can be made invariant to rescaling, rotation or translation of the input data by constraining the weights to meet certain requirements [Giles 1988]. This is similar in some respects to the sub-net weight sharing used by the best standard model net for the E-task. Thus although the present results are not optimistic the case for higher order nets is still open.

Without the power of genetic search for network architecture optimisation our confidence in these conclusions would be substantially reduced. The evolutionary search process, although computationally expensive, offers a powerful tool to explore alternative network designs.

Acknowledgement: This work was supported by SERC grant GR/G 18391.

References.

[Bimbot 1990] F. Bimbot. *Speech processing and recognition using integrated neurocomputing techniques*. Technical report BRA3228, Cap Gemini Innovation, 118 rue de Tocqueville, 75017 Paris, France.

[Dodd 1991] N. Dodd, D. Macfarlane and C. Marland. *Optimisation of artificial network structure using genetic techniques implemented on multiple transputers*. TRANSPUTING 91, Vol 2 [Eds D.Styles, T. Kunii, A.Bakkers], IOS Press 1991.

[Giles 1988] C. Giles, R. Griffin and T. Maxwell. *Encoding geometric invariances in higher order neural networks*. Neural Information Processing Systems, American Institute of Physics, New York, pp 301-309, 1988.

[Jones 1992] Antonia J. Jones. *The modular construction of dynamic nets*. To appear in: Neural Computing

Applications Journal, 1992.

[Jones 1992] Antonia J. Jones. *Genetic algorithms and their applications to the design of neural networks*. To appear in: Neural Computing Applications Journal, 1992.

[Macfarlane 1990] D. Macfarlane and I. East. *An investigation of several parallel genetic algorithms*. OUG-12: Tools and Techniques for Transputer Applications. IOS Press, 1990.

[Shepard 1958] R. N. Shepard. *Stimulus and response generalisation: Deduction of the generalisation gradient from a trace model*. Psychological Review 65, 242-256, 1958.

[Shepard 1987] R. N. Shepard. *Towards a universal law of generalisation for psychological science*. Science 237, 1987.

[Tawel 1989] R. Tawel. *Does the neuron "learn" like the synapse?* Advances in Neural Information Processing 1, pp 169-176. Ed. D. S. Touretzky, Morgan Kaufmann, 1989.