A Dynamic Neural Network Architecture by sequential Partitioning of the input space[1].

R.S.Shadafan[2] & M.Niranjan
May 13, 1993

Cambridge University Engineering Department,
Trumpington St., Cambridge, CB2 1PZ, England.
Email: rss/niranjan@eng.cam.ac.uk

[1]Part of this work was published in the proceedings of the IEE International Conference on Neural Networks, March 28 - April 1, 1993, San Francisco, Ca., vol1, pp. 226-231.
[2]Supported by a grant from Trinity College, Cambridge, and Karim Rida Said Foundation.

Abstract

We present a sequential approach to training multilayer perceptron for pattern classification applications. The network is presented with each item of data only once and its architecture is dynamically adjusted during training. At the arrival of each example, a decision whether to increase the complexity of the network, or simply train the existing nodes is made based on three heuristic criteria. These criteria measure the position of the new item of data in the input space with respect to the information currently stored in the network.

During the training process, each layer is assumed to be an independent entity with its particular input space. By adding nodes to each layer, the algorithm effectively adding a hyperplane to the input space hence adding a partition in the input space for that layer. When existing nodes are sufficient to accommodate the incoming input, the involved hidden nodes will be trained accordingly.

Each hidden unit in the network is trained in closed form by means of a Recursive Least Squares (RLS) algorithm. A local covariance matrix of the data is maintained at each node and the closed form solution is recursively updated. The three criteria are computed from these covariance matrices with minimum computational cost.

The performance of the algorithm is illustrated on two problems. The first problem is the two dimensional Peterson & Barney vowel data. The second problem is a 32 dimensional data used for wheat classification. The sequential nature of the algorithm has an efficient hardware implementation in the form of systolic arrays, and the incremental training idea has better biological plausibility when compared with iterative methods.

Contents

1 Introduction

Neural Networks have been successfully applied to many pattern recognition problems. Usually, the classification problem is cast as an interpolation problem by assigning real valued 'targets'. The network attempts to interpolate these targets at locations defined by a set of training data in multi dimensional input space or feature space. In this input space, the network may be seen as approximating a nonlinear function. Often, a contour on the the network output function is treated as a discriminant function. Sometimes, with proper normalisation, network outputs are also interpreted as posterior probabilities of class membership. Training, or parameter estimation, is usually done by minimising the total squared interpolation error over the collection of training examples, using a gradient descent type of procedure.

In this paper, we consider an algorithm that is suitable for data that arrive sequentially. As each item of data arrives, the classifier has to be trained incrementally so as to satisfy the new arrival and at the same time be consistent with the past observations. Past observations are not retained. Additionally, even in situations where all data are available before training, one might be able to achieve a computational advantage with sequential learning procedures as such algorithms see each item of data only once.

A further motivation for this work is the observation that in a multi-layer perceptron classifier the contribution of each unit in hidden layers is essentially 'local'. If we view the classification boundary as being approximated by segments of hyperplanes, the positions of these segments are determined primarily by data that lie close to them. Data far away from the segment boundaries contribute little to the error function. In this paper, we exploit this observation by sequentially partitioning the input space so that training nodes in the multi-layer perceptron is local.

We present to the network each example only once and dynamically increase the size of the network. As every example arrives, the net will decide whether to add a new node or simply train the existing nodes according to some criteria.

There are many advantages in this approach, the network can be expanded to a multi-layer network by introducing new layers whenever they are needed, training them separately and sequentially with the set of outputs from the previous layer. In addition, this network allows continuous learning (*i.e.* learning every new example even outside the training set), which means the net will dynamically forget and remember examples according to their distributions and occurrences. Such on-line learning is especially useful when the data arrive sequentially and have a slowly drifting pattern of statistical behaviour.

Every node in the network is trained by the Recursive Least Squares (RLS) algorithm, a technique which is widely known in the area of adaptive filters [1]. Azimi-Sadjadi *et al.* [2] and Sin *et al.* [3] have used RLS algorithm in training Multi-Layer Perceptrons (MLP). Other related work is the use of the Extended Kalman Filter (EKF) algorithm, which is similar in form to RLS, but allows one to incorporate knowledge or estimates of noise variances in the data. Singhal *et al.* [4] have used the EKF algorithm to train an MLP. The EKF algorithm has also been used for estimating Radial Basis Functions model by Kadirkamanathan *et al.* [5] & [15]. The common theme underlying these methods that we borrow from adaptive signal processing is that the inverse of a matrix appearing in the closed form expression for the least squares solution of a system of equations can be evaluated recursively, using the matrix inversion lemma.

In the following sections we will show how a perceptron can be trained by the RLS method, and construct the criteria which will control network growth. Then, the sequential machine will be introduced and illustrated by a simple example. Finally, we will demonstrate the performance of this classifier on two real life problems, namely the vowel classification as a low 2-dimensional problem, and the wheat classification as a large 33-dimensional problem.

2 Recursive Least Squares

Given a set of N training data $\mathbf{x}_i,\ t_i,\quad i = 1, ..., N$, where $\mathbf{x}_i \in \mathcal{R}^{p+1}$and t_i are real values known as targets, the interpolation conditions for a perceptron model are a set of simultaneous equations given by,

$$t_i \;=\; f(\mathbf{x}_i^t\ \mathbf{w}_i), \quad i = 1, ..., N \tag{1}$$

For input data of dimension p, we work in a $p+1$ dimensional space to absorb the bias term in the perceptron model, making the notations easier. The $(p+1)^{\text{th}}$ component is set to 1. The nonlinear function $f(.)$ is usually a sigmoid, given by

$$f(\alpha) \;=\; \frac{1}{1+\exp(-\alpha)} \tag{2}$$

The system of equations becomes,

$$\mathbf{x}_i^t\ \mathbf{w}_i \;=\; \log \frac{t_i}{1-t_i} \tag{3}$$

In matrix notation, we rewrite this as

$$X\ \mathbf{w} \;=\; C, \tag{4}$$

where X is a Nx$(p+1)$ matrix of the input data vectors, $\mathbf{w}$ is the unknown parameters of the perceptron and C is computed from the target values by the inverse of the nonlinearity. Due to the nature of the log function, for pattern recognition problems we use 0.1 and 0.9 as targets, instead of 0 and 1.

The above is usually an overdetermined set, *i.e.* $N >> (p+1)$, of linear equations. The least squares solution that minimises the total squared error,

$$E \;=\; \frac{1}{2}\sum_{i=1}^{N}(e_i)^2, \tag{5}$$

$$e_i \;=\; t_i - f(\mathbf{x}_i^t\ \mathbf{w}_i), \tag{6}$$

can be obtained from the pseudo-inverse as,

$$\mathbf{w} \;=\; [X^t\ X]^{-1}\ X^t\ C, \tag{7}$$

Letting $[X^t\ X] = B$ and $X^t\ C = G$, reduces this to:

$$\mathbf{w} \;=\; B^{-1}\ G, \tag{8}$$

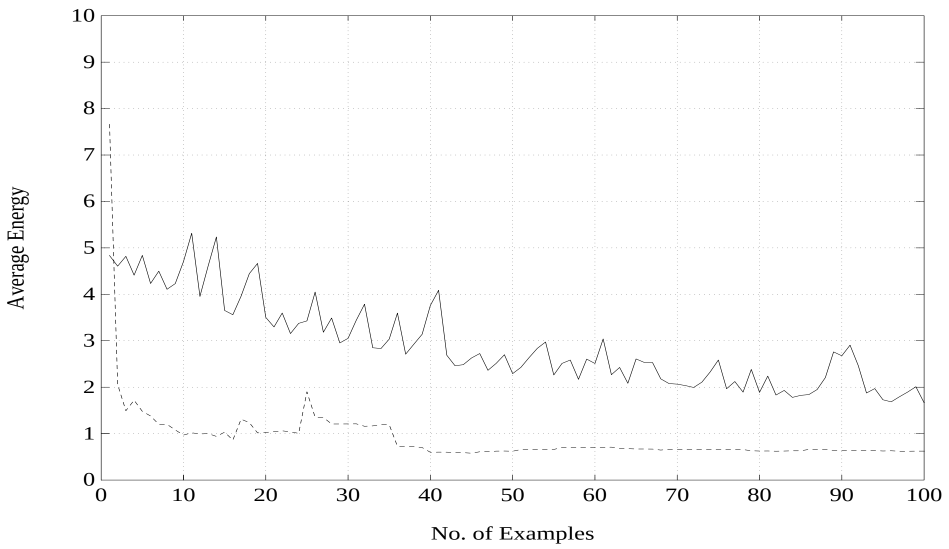

Figure 1: A comparison on a simple vowel classification problem between the LMS (solid line) and the RLS (dashed line) convergence rates as a function of number of examples the network was sequentially presented with. The problem here is linearly separable and has only two dimensions.

Here, the matrix B is the auto-correlation matrix of the input patterns, and the vector G is the cross-correlation between the inputs and their designated targets [10], both quantities can be found recursively at the introduction of a new example $\mathbf{x}_i$ as follows:

$$B_i = B_{i-1} + \mathbf{x}_i\ \mathbf{x}_i^t, \tag{9}$$
$$G_i = G_{i-1} + \mathbf{x}_i\ t_i, \tag{10}$$

In sequential training, we use the matrix-inversion lemma [1], to compute the inverse of these terms recursively. The inverse of B_i can be computed from

$$B_i^{-1} = B_{i-1}^{-1} - \frac{B_{i-1}^{-1}\ \mathbf{x}_i\ \mathbf{x}_i^t\ B_{i-1}^{-1}}{(1 + \mathbf{x}_i^t\ B_{i-1}^{-1}\ \mathbf{x}_i)}. \tag{11}$$

For sequential learning we prefer the above RLS approach to a gradient descent type algorithm. This is because the network sees each example only once and additional information is retained in the form of inverse covariance matrices. A simpler approach is to use a Widrow-Hoff type algorithm (the Least Mean Squares (LMS) algorithm in adaptive signal processing) to perform an approximate gradient descent using $\partial(e_i)^2/\partial\mathbf{w}$ at the arrival of each example $\mathbf{x}_i$, t_i. In this case, the information we extract from each item of data is minimal. With the RLS algorithm, we accumulate data covariance information in addition to the gradient information. Though the amount of computation at each iteration is much higher than the LMS type approach, convergence is much faster, see Fig. 1. Further, in a practical implementation where the data arrival is sequential (*e.g.* time series prediction or on-line control), we can pipeline these computations on a systolic array type architecture [6, 7].

3 The Sequential Machine

We now look at how to incorporate the use of the RLS algorithm to train a single perceptron into building a multi-layer network of perceptrons sequentially. This is done essentially by partitioning the input space of each layer into local areas and the application of the RLS algorithm on each node local to the newly presented datum. As each example arrives, we apply three criteria to either associate the example with an existing node and re-estimate its parameters or expand the network if the existing nodes cannot properly account for the data.

3.1 Criterion for linear separability

This criterion will check whether our new input example will cause an untrainable (or less comfortable training) situations with every node in the current machine. This is vital to detect since unrelaxed training using the RLS-algorithm will cause a bad placement of the hyperplane separating the two classes for any node in the network.

This criterion is based on the correlation between the outputs $Y = f(X\ \mathbf{w})$ and their designated targets T, (the quantity $T^t\ Y$). To make use of the quantities B_i^{-1} & G_i computed by the RLS algorithm, we consider the approximate corresponding linear version of the same quantity $z = C^t\ X\ \mathbf{w}$. If the RLS solution was exact, which is the case when $X\ \mathbf{w} = C$, (hence $Y = T$), then z will go to a maximum value of $C^t\ C$. On the other hand, if the RLS solution for $\mathbf{w}$ totally failed, the correlation between $X\ \mathbf{w}$ and C is zero.

The above idea can be developed to a criterion figure which will equal to one when the network has a total success and to zero if the network has a total failure, when training the new example with the existing B^{-1} & G. This can be expressed in the normalised correlation:

$$Z = \frac{C^t\ X\ \mathbf{w}}{C^t\ C}, \tag{12}$$

$$= \frac{G^t\ B^{-1}\ G^t}{C^t\ C}. \tag{13}$$

The normalised quantity Z can take any value between 0.0 (least correlation) and 1.0 (most correlation), the degree of correlation depends mostly on whether the new example maintains the linear separability in the input space to the boundary under test. We can set a threshold value $Z_\circ \epsilon [0.0, 1.0]$ to specify the permitted degree of linear separability needed to be maintained to allow training.

For every example $\mathbf{x}_i$ presented to the system, B_i^{-1} & G_i for each node will be temporarily updated using equations 10&11, and Z can be calculated using equation 13. If $Z > Z_\circ$ for a particular node then training this example with that node is going to keep the situation linearly separable and B_i^{-1} & G_i will keep their present values, otherwise training is not relaxed or the new example is linearly inseparable therefore a new node is introduced to ease the training process, and B_i^{-1} & G_i will retain their previous values. Fig. 2 shows different values of Z for a newly presented example from both classes around the boundary with respect to its location in the input space.

The Threshold $Z_\circ$ is set according to how relaxed the user wants the network. However, more relaxed network will create more nodes hence larger network size.

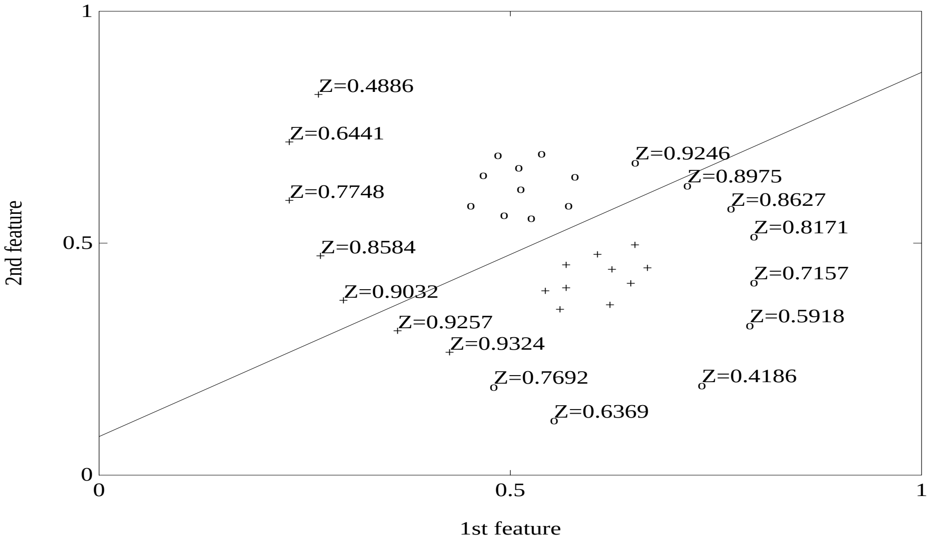

Figure 2: A system with two classes in two dimensional space seperated by one hyperplane. New examples are introduced from both classes at different location labeled with their corresponding Z value.

3.2 Criterion for remoteness

There might be cases where the new example is linearly separable (*i.e.* high Z value) however it is too remote to be associated with its class cluster. In this case, we associate the input to a new cluster of the same class. To measure remoteness, we introduce a second criterion, again determined by quantities that can be computed by the RLS algorithm.

For instance we can compute the quantities B^{-1} & G for each class the node is classifying, then each new examples will be trained with the same class but with opposite target using the RLS algorithm for a new hyper plane. The Error $|c_i - \mathbf{x}_i\, W|$ will be an indication of how near $\mathbf{x}_i$ is to that cluster, where c_i is the corresponding inverse of the nonlinearity of the target t_i. This value will swing from near zero to $|2\ c_i|$ as $\mathbf{x}_i$ travels from remote distance to the centre of that cluster. Normalizing this quantity will give a measure of how remote this input is to that cluster:

$$D \quad = \quad \frac{|c_i - \mathbf{x}_i\ \mathbf{w}|}{|2\ c_i|}. \tag{14}$$

The advantage of this criterion is that it takes the geometry of the cluster tested into consideration by applying the RLS algorithm on that cluster. However, separate B^{-1} & G quantities for each side of the hyperplanes must be maintained and calculated each time the node has been trained with new input. Fig. 3 shows different values of D for newly introduced examples to a system of two classes, in each case D is computed for both classes and the lowest value of the two is chosen.

3.3 Criterion for locality

Locality in this context is defined as follows:

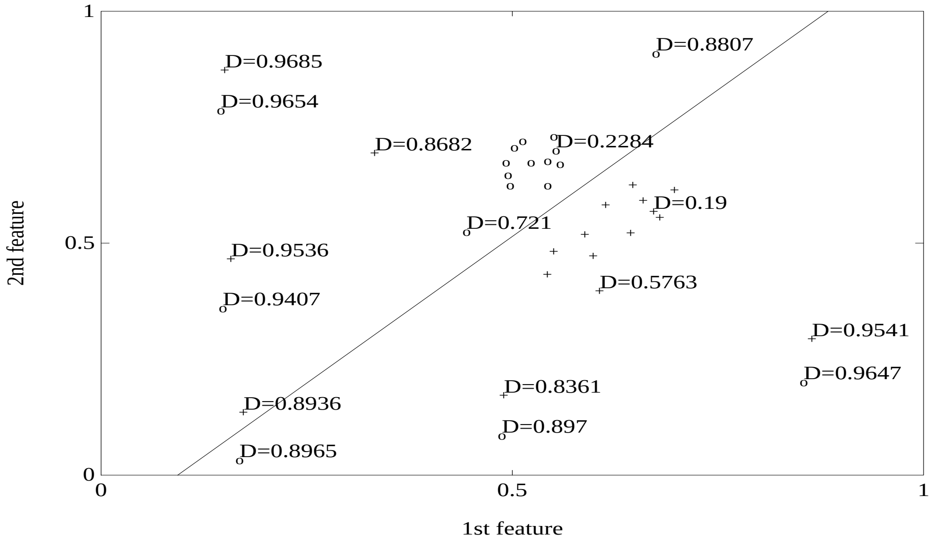

Figure 3: A two class system similar to Fig. 1. The target of the new example does not affect D since each example will be trained with oppsite target of the cluster under test. D is calculated for both classes and the lower value is always chosen.

The input $\mathbf{x}_i$ *is local to a boundary if there is no other boundary between the location of the input and that boundary.*

When testing the present input for locality to a boundary, the input is orthogonally projected on the boundary producing a new point $\mathbf{x}_i^{'}$ in the input space which will be tested by all the other nodes in the same layer. The present example is local only if all the outputs of the other nodes maintained their status within the same class.

Since the weight vector $\mathbf{w}$ is always perpendicular to the boundary, the difference between $\mathbf{x}_i^{'}$ and $\mathbf{x}_i$ should be in the same/opposite direction, or:

$$\mathbf{x}_i^{'} - \mathbf{x}_i \quad = \quad q\ \mathbf{w}, \tag{15}$$

where q is a scalar, and all the vectors are taken without the bias term in this equation. Moreover, $\mathbf{x}_i^{'}$ is on the boundary, hence:

$$\mathbf{x}_i^{'}\ \mathbf{w} = 0, \tag{16}$$

where vectors now include the bias term. Using these two equations we can solve for $\mathbf{x}_i^{'}$ and carry out the test on the other nodes. Changing output status for a node k can be detected when the quantity $\mathbf{x}_i^{'}\ \mathbf{w}_k$ has a different sign than $\mathbf{x}_i\ \mathbf{w}_k$. Computationally, this can be done by multiplying both quantities:

$$L_k \quad = \quad [\mathbf{x}_i^{'}\ \mathbf{w}_k][\mathbf{x}_i\ \mathbf{w}_k], \tag{17}$$

if the answer is negative then a change in status has occurred, L_k is positive otherwise.

3.4 The final classifier and an illustrative example

The machine developed here is to classify only two classes, but expansion to more than two classes is straight forward. For r class problems, we will build r machines each will classify one class against the other $r-1$ classes, hence the final classifier will have r outputs, each will map to only one class. In a two class problem, the machine will only have one output which fires when the presented example belongs to one of the classes.

Layers generated within the network will be treated as an entity and trained separately, nodes will be added to any layer to partition the input space of that layer to solve for nonseparable (or unrelaxed) training situations.

The algorithm for partitioning the input space is summarised as follows, snap shots of the process are illustrated in Fig. 4:

1. Start the machine with one node, and train it with the first example using the RLS algorithm.
2. Present a new example and for every node in this layer:
   - Check remoteness by computing D for both sides of the boundary
   - Check locality by computing L.
   - Check separability by computing Z.
3. If the example is remote to all boundaries, create a new node and train it with this example using the RLS algorithm, go back to step 2.
4. If the example is local and linearly separable with a set of nodes, train these nodes with the example using the RLS algorithm, go back to step 2.
5. If the example is local but linearly nonseparable with a node, then create a new node and use B^{-1} & G of the opposite class of that node and the present class to train the newly created node, go to step 2.

The addition of more layers will be controlled by the number of outputs of the final layer, if the final layer developed to have more than one output, another layer with one node will be added, so the machine will always has one final output all the time. The partitioning algorithm will then be applied to the new output layer with its input being fed from the previous layer. For each example presented, the process will be repeated until the machine converges to one output. The behaviour of the model is shown in Fig.5, the network eventually converged into 2 layers with one node in the output layer.

The quantities needed to be preserved in some memory are B^{-1} & G for the RLS algorithm. Each node will need B^{-1} & G for both classes combined to train the node under test, and other two B^{-1} & G for each class separately to be used for training newly created nodes. It will also need to keep track of the quantity $C^t\,C$ for both classes combined to normalize Z, and $C^t\,C$ for each class to be deported to the newly born nodes when needed. These quantities consumes relatively less memory when compared to other neural network techniques. Memory size here does not depend on the number of training examples rather than the size of the network, as both G, which is a vector, and B, which is a square matrix, do not grow until a new node is added to the same layer.

One fact to point out is that since network size and its parameters are modified sequentially with every example without anticipating future examples, the final network architecture depends on the sequence of how the examples are presented to the network during training process, this might affect the final performance of the network. However, if the number of examples is large enough for the network architecture to converge into

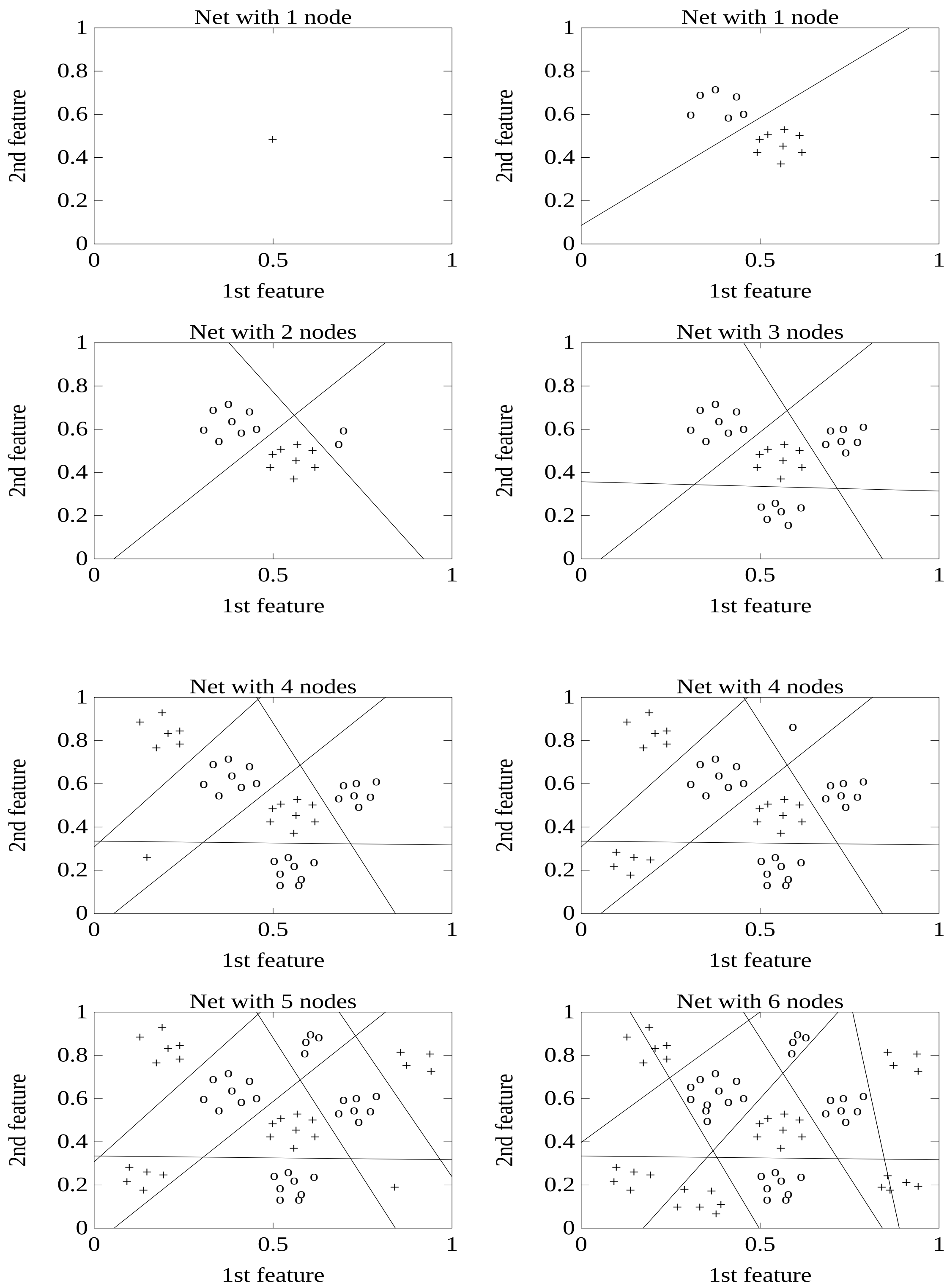

Figure 4: Space partitioning for one layer of 2-D input space. New hyperplanes are introduced by creating new nodes when neccessary. Old hyperplanes are trained when training is relaxed enough. Both training and creating is done upon the arrival of a new example.

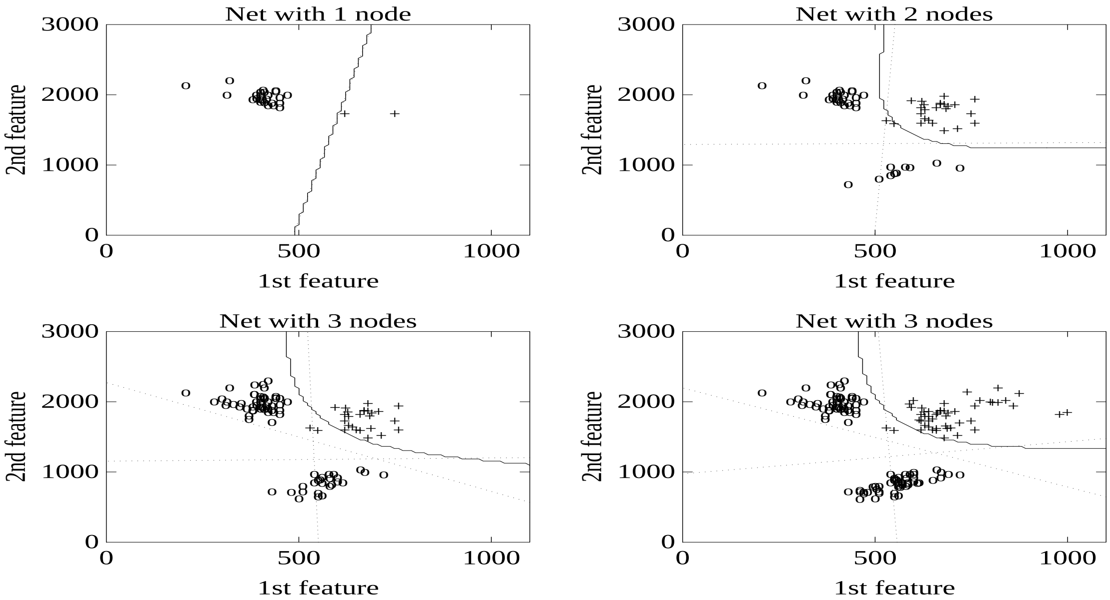

Figure 5: Simulation of the algorithm on a simple 2-D problem. The dotted lines denote the class boundaries contributed by each of the three nodes in the hidden layer, the solid line is the effective class boundary of the whole network.

a proper solution, then its performance is expected to have small variations for different sequences. We will show that this is true for the vowel data, (see Fig. 8).

Moreover, one might anticipate that noisy examples will shape the network to a poorer solution, but because of the localisation of training and the stability of the RLS learning rule, the network maintained a good generalization, although it affected the network size, as noise added to the examples will expand the input space and introduce more complexity to the final boundary shape. This point is illustrated on one of the vowel classes in section 4.1, where we added noise to the original training data and examined the final architecture and its performance on clean testing set of data, (see Fig 9).

4 Experimental Work

We now show the performance of the model on two pattern recognition problems. The first is a 2-dimensional vowel recognition problem. The second is a more complicated 33-dimensional problem involves wheat classification.

4.1 Vowel Recognition

The data here is based on Peterson & Barney database [14]. The classes are vowel sounds characterised by the first four formant frequencies made by people of different ages and gender. To visualize the problem, we reduced the problem to two features by taking the first two formant frequencies.

The database consists of 1494 examples. It was randomly mixed without repeating any example, and split into two set: a 1000 example set used as the training examples and a 494 example set used as the testing examples.

The examples are mapped to 10 vowels (10 classes). Therefore, we built ten one-output machines, each machine has a hidden layer and one output which fires in response to only one specific class. The hidden layer was allowed to evolve to a number of nodes enough to partition the input space using our algorithm. Table 1 shows the performance of each machine and its final size . These results also indicate good generalization to the testing data set.

Vowel	IY	IH	EH	AE	AH	AA	AO	UH	UW	ER	Ave.
Train	97.6	95.5	94.4	96.8	95.3	96.0	95.0	93.4	95.9	92.8	95.3
Test	98.0	96	94.3	96.4	94.7	97.0	95.5	91.5	94.7	91.3	94.9
Net size	9	19	16	16	41	17	15	15	11	28	18.7

Table 1: Results of the vowel problem. Each percentage corresponds to individual class recognition. Net size corresponds to number of nodes in the hidden layer.

Further, the outputs of the above 10 networks were tested collectively to recognise the corresponding classes. The scheme was to choose the highest output to designate the correct class. Accordingly, the networks managed to correctly classify 78% of the training data, and 74.5% of the testing data. This modest result can be due to the overlapping between classes, taking into account only the first two formant frequencies were considered in the training data, see Fig. 6. Nevertheless, our results are comparable to other sequential and non-sequential methods applied on the same problem, reported in [15, 16, 17]. Table 2 shows these reported results and the results of our algorithm. It has to be noted that the work of Kadirkamanathan [15] is the nearest to our work in terms of the size of the training and the testing sets, Lowe [16] used only 338 examples as the training set and 333 examples as the testing set. Other results - not mentioned here - were reported by Prager [19] and Jacobs *et al.* [18], however, Prager used the first four formant frequencies instead of two, and Jacobs only used four out of the ten vowel classes. Overall picture of how the classes were finally bounded using our scheme is shown in Fig. 7.

Classifier	Train	Test
Optimum Linear Transformation [16]	41.12	40.24
Distance-to-class mean (Euc.) [16]	68.05	68.77
Distance-to-class mean (Mah.) [16]	78.40	80.18
Full Gaussian Classifier [16]	76.63	74.85
Nearest Neighbour [16]	100.0	77.48
K-Nearest Neighbour (K=5) [16]	82.25	81.38
Gaussian RBF (36 hidden) [16]	na	80.18
Thin plate spline (32 hidden) [16]	na	81.08
Gaussian RBF (softmax) [17]	na	78
Dynamic network (85 hidden) [15]	76.58	75.40
Our Sequential space partitioning	78.0	74.5

Table 2: Percenatge correct of different reported classifiers and our classifier. na: not available.

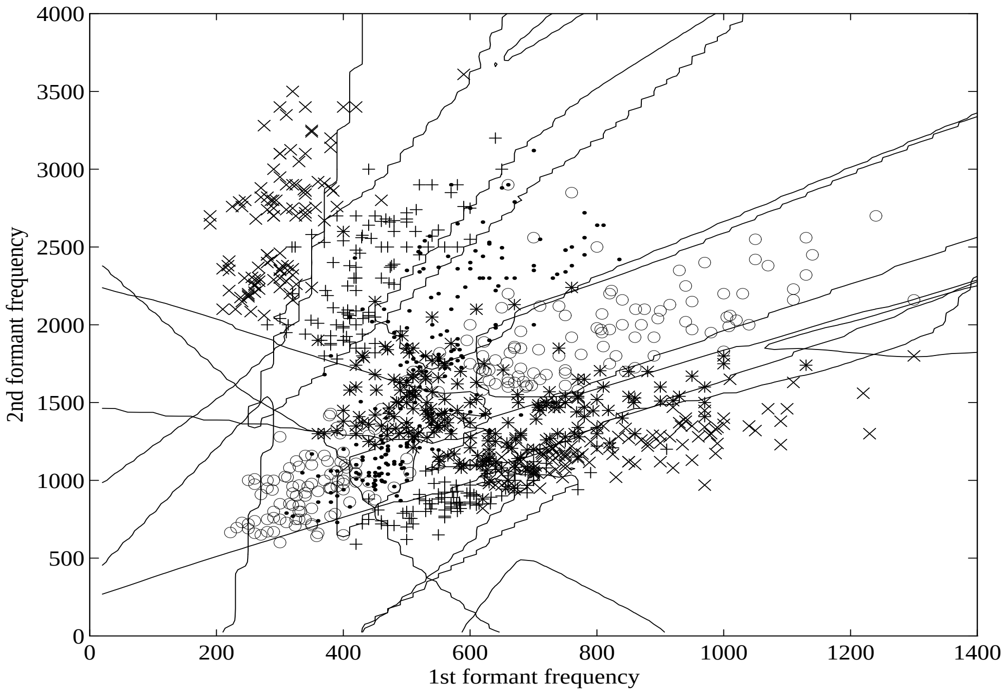

Figure 6: Initial boundaries made by the 10 networks at the output level 0.5, notice the overlapping.

To illustrate the effect of the input data sequence on the architecture and its final performance, we trained one of the machine with different sequences by randomly changing the sequence of the first 250-300 examples in the training set without any repetition in the examples, as these examples constitute the most crucial factor on shaping the network. The dynamics of the hidden layer for different sequences for the vowel UH is shown in Fig. 8. We chose this vowel because it is a difficult problem as noticed from its low performance and it is surrounded by other classes from all direction, see Fig. 7. Although different sequences resulted in different network sizes, the performances, on both training and testing sets, were comparable. Table 3 shows the performances for 10 different sequences and the corresponding hidden layer size.

cycles	1	2	3	4	5	6	7	8	9	10	Ave	Std. Dev.
train	93.9	93.7	93.8	93.8	93.0	93.5	93.3	93.4	93.8	93.3	93.6	0.3
test	91.7	91.9	91.3	90.9	91.7	91.7	89.7	91.3	92.1	90.9	91.3	0.7
net size	15	16	31	12	13	14	21	12	18	13	16.5	5.8

Table 3: Percenatge correct results of 10 different training input sequence of the vowel (UH). Net size corresponds to the number of nodes in the hidden layer.

On the noise issue, again we selected the vowel (UH) data and added different levels of noise to the training examples, and tested its performance on the clean testing set. During these experiment, hidden layer size was not allowed to exceed 100 units. The results are shown in Fig 9 for both normal and uniform added noise. The network size

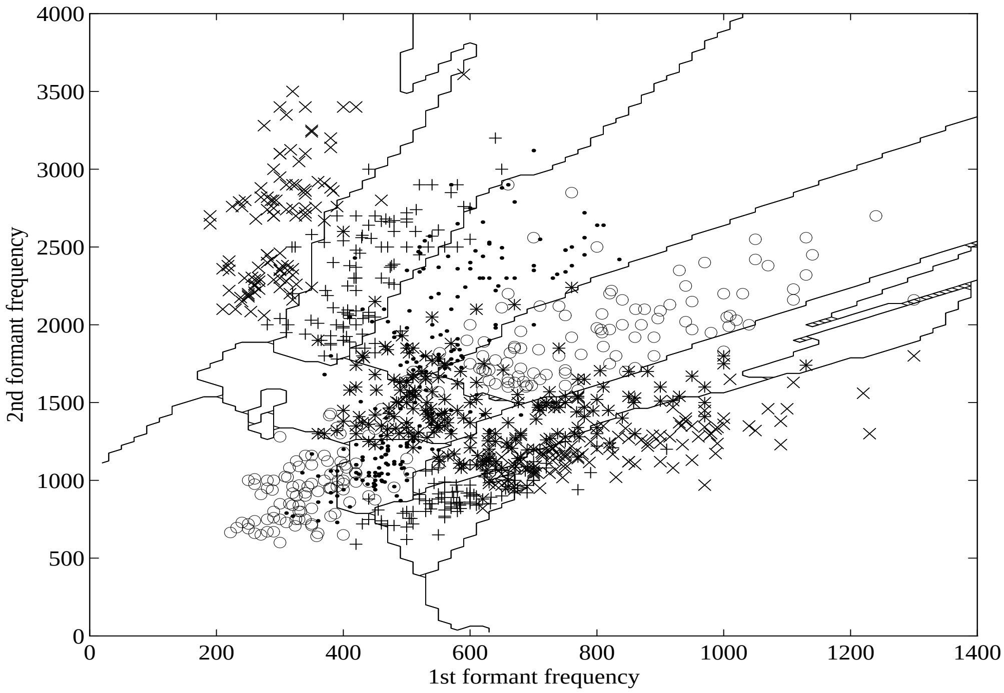

Figure 7: The final classification of the 10 classes in the vowel problem using the *The correct class at the highest output* scheme. Class (UH) is denoted by '.' in the midle of the graph.

increased with the noise level, but the performance was maintained on both training and testing sets.

4.2 The Wheat problem

The wheat problem constitute a higher dimension database. The task here is to classify one strain wheat grain called mercia against 21 other varieties. Mercia wheat grain is important for good quality bakery. The database is supplied by a special imaging system developed at the National Institute for Agricultural Botany in Cambridge [12]& [13] for wheat identification. It contains 23006 training examples and a separate set of 2000 examples for testing. Each example is a vector of 33 elements.

Out of the entire training set we selected 2000 examples (1151 mercia and 849 other classes) as our training set. The experiment involved three networks using three different $Z_\circ$ values see section 3.1, leading to different network sizes, but all made to converge to one node in the second layer. The network sizes and their performances are summarised in table 4

The same problem was tackled by Prager [19] using different kinds of training methods, for the sake of comparison the performances of these networks are listed in table 5. Our results put us in the middle amongst Prager's networks, however, emphasis is drawn to the number of data cycles each network needed to converge compared to one data cycle in our case.

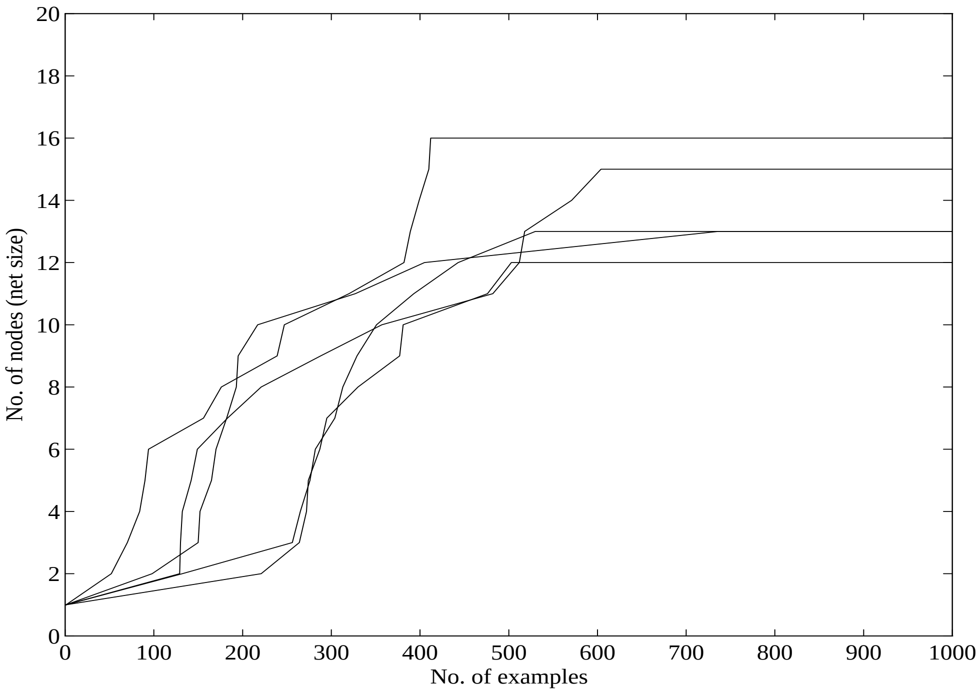

Figure 8: Growth patterns of a hidden layer for the class (vowel UH) during training for different presentation sequences of the input examples.

$Z_\circ$	0.9	0.45	0.35
Net Size	17	3	1
Train-2000	81.7	83.2	84.2
Train-23006	79.1	80.9	81.9
Test-2000	82.1	83.5	84.5

Table 4: Results of the wheat problem.

5 Conclusions.

A sequential approach to designing a Multi-Layer Perceptron pattern classifier is presented. The architecture of the MLP (*i.e.* the number of nodes) is increased dynamically with the complexity required to classify the incoming data. Training each node in the network is done in closed form using the Recursive Least Squares algorithm with **local** covariance matrices. Thus each node contributes to a segment of class boundary that makes local classification decisions. The approach has the advantage that training is sequential hence it is fast and particularly useful when the nature of the problem is essentially sequential too. Whereas, iterative methods require to have the whole batch of the training set and pass it to the network for a large number of times, making it slower and harder to implement. In situations where there is a large number of data items, the computations may be pipelined and implemented on a systolic array type architecture. We are currently working on a simulation of such type of an architecture.

The performance of the algorithm is illustrated on the simple two dimensional vowel

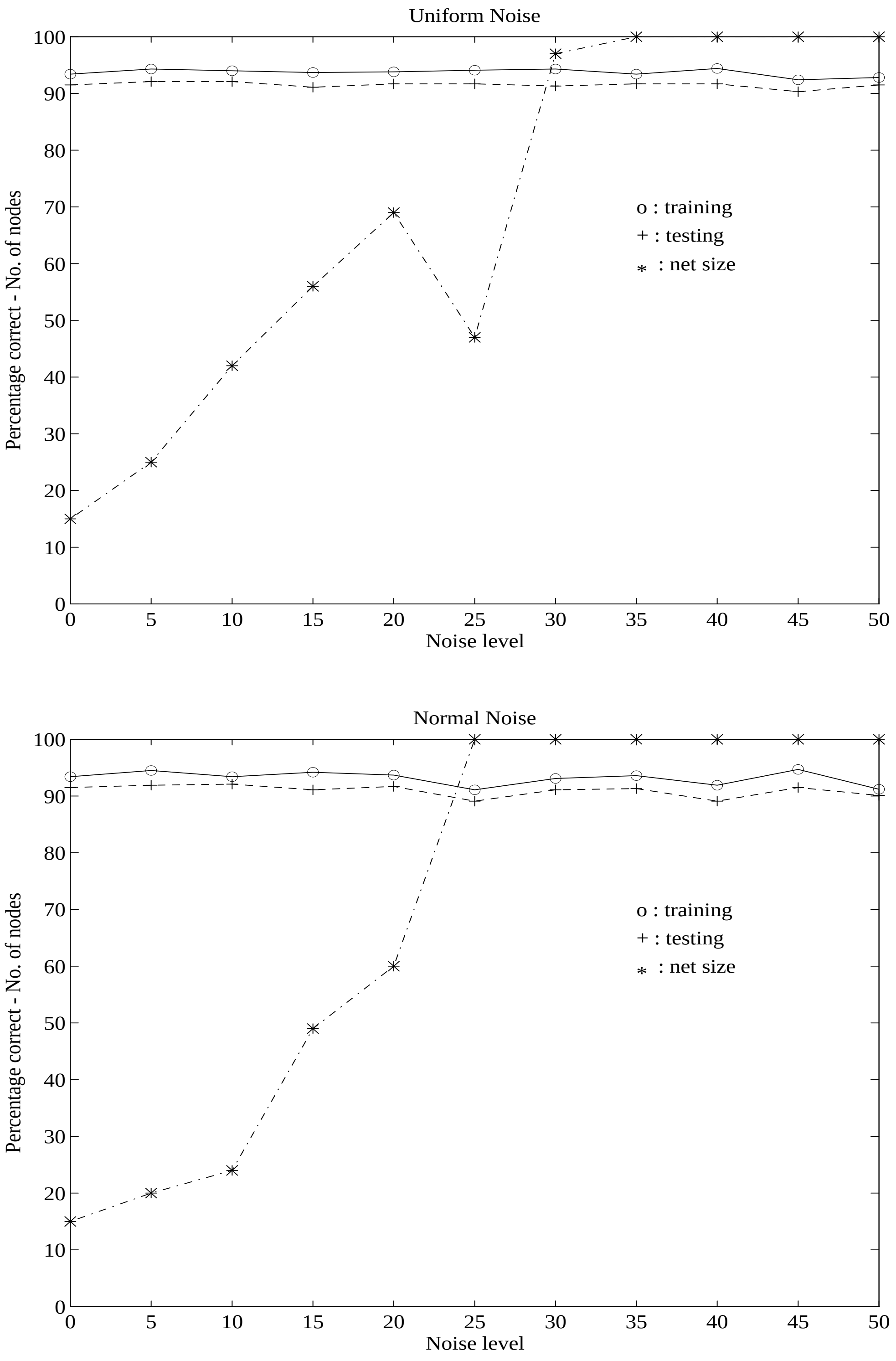

Figure 9: Performance of the algorithm under noisy environment. The problem used is the vowel (UH). The y-axis represents percentage correct for training and testing data, it also represents the number of nodes in the hidden layer. At any time, the number of nodes was not allowed to exceed the 100 unit limit.

Network type	Training Cycles	Train	Test
Single Layer Network:	42818	83.3%	86.9%
500 Random Locations:	3714	72.8%	77.4%
Approx. 500 pruned Locations:	3476	78.4%	82.8%
Sequential Space Partitioning:	1	81.9%	84.5%

Table 5: Performance of Prager's Networks on the wheat problem and our network performance.

classification and on the larger dimension Wheat problem. The results are comparable to other methods on the same benchmark problems, although less computations were required. As expected, the order of the input data effected the network architecture, but, provided that the training data set is large enough, these effects can be minor and may have negligible effect on the final performance and generalization of the network. They also show encouraging signs to maintain performance even under noisy environment, although the increase in complexity of the noise data increased the size of the network. We believe that this increase in the net size is not contributing much to its performance, and many of these created boundaries are redundant. We are currently working on a pruning algorithm to remove such redundant units.

References

[1] Simon Haykin, *Introduction To Adaptive Filters.* Macmillan Publishing Company, 1984.

[2] M. R. Azimi-Sadjadi and S. Sheedvash, "Recursive node creation in back-propagation neural networks using orthogonal projection method," *ICASSP 91*, Toronto.

[3] S-K. Sin and R. J. P. deFigueiredo, "An evolution-oriented learning algorithm for optimal interpolative net," *IEEE Trans Neural Networks*, vol 3, No 2, March 1992.

[4] Sharad Singhal and Lance Wu, " Training feed-forward networks by the extended Kalman algorithm," *ICASSP 89*, Glasgow.

[5] V. Kadirkamanathan and M. Niranjan, " Nonlinear adaptive filtering in nonstationary environments,", ICASSP 91, Toronto.

[6] J. V. McCanny and J. C. White. *VLSI Technology and Design.* Academic Press, 1987.

[7] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, 1988.

[8] C. Chatfield and A. J. Collins, *Introduction To Multivariate Analysis.* Chapman and Hall, pp. 190-193, 1980.

[9] N. R. Draper and H. Smith, *Applied Regression Analysis.* John Wiley & Sons, Inc. 1966.

[10] William A. Gardner, *Introduction To Random Processes*. Macmillan Publishing Company. 1986.

[11] Nils Nilsson, *The Mathematical Foundations of Learning Machines.* Morgan Kaufmann, 1990.c

[12] P. D. Keefe and S. R. Draper, " The measurement of new characters for cultivar identification in wheat using machine vision," *Seed Sci. and Technol.* , vol 14, pp. 715-724, 1986.

[13] P. D. Keefe, " Observations concerning shape variation in wheat grains," *Seed Sci. and Technol.* , vol 18, pp. 629-640, 1990.

[14] G. E. Peterson and H. L. Barney, "Control methods used in a study of the vowels," *JASA*, vol 24, pp. 175-184, 1952.

[15] V. Kadirkamanathan and M. Niranjan, "Application of an architecturally dynamic network for speech pattern classification," *Proc. of the Inst. of Acoustics*, vol 14, part 6, pp 343-350, 1992.

[16] D. Lowe, "Adaptive radial basis function nonlinearities and the problem of generalisation," *Proc. IEE Conference on Artificial Neural Network*, 1989.

[17] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relation to statistical pattern recognition," F.Fougelman-Soulie & J. Herault (eds.) *Neuro-computing: algorithms, architectures and applications,* Springer-Verlag, 1990.

[18] R. Jacobs, M. Jordan, S. Nowlan, G. Hinton, "Adaptive Mixtures of local Experts," *Neural Computation,* vol 3, No 1, spring, 1991.

[19] Richard W. Prager, " Some experiments with fixed non-linear mappings and single layer networks," Technical Report (CUED/F-INFENG/TR.106), Cambridge University Engineering department, July, 1992.