Adaptive Neural Tree for Pattern Recognition *

T. Li Y.Y. Tang C.Y. Suen
Department of Computer Science
Concordia University
1455 de Maisonneuve Blvd. W., Montreal, Canada H3G 1M8

L.Y. Fang A. Jennings
Artificial Intelligence Systems Section
Telecom Australia Research Labs
770 Blackburn Road, Clayton VIC, Australia 3168

Abstract

This paper presents an adaptive self-organizing neural tree and its application to character recognition. The neural tree is suitable for hierarchical classification and it can grow and shrink to adapt to the changing environment. It also makes parametric adaptation to cope with small changes in the environment. When applied to character pattern recognition, it shows promising performance.

1 Introduction

Tree-structured classifiers [2, 3, 6, 15, 21, 27] have been widely used in pattern recognition and have demonstrated competitive performance. Traditional approaches are not able to adapt to changing environments. Sometimes, both parametric adaptation and structural adaptation may be needed to achieve satisfactory performance. The neural network approach [9, 12, 18] provides a natural way to adaptive processing. We have developed a neural tree architecture for classification and vector quantization [5, 17]. This neural architecture has shown competitive performance as compared with other neural networks and traditional approaches. In this paper, we present an adaptive neural tree architecture and its application in character recognition. The adaptive neural tree is efficient in storage requirement and computation time.

Adaptive neural networks which grow when new concepts are learned and shrink when old concepts are disposed can be very useful. Such neural networks can be used very efficiently. The initial training phase can also be eliminated for certain applications. A few adaptive neural networks with supervised and unsupervised learning capability [1, 4, 16, 28] have been proposed. However, the architecture in [16] is the only one which can follow the signal source while the others only grow during learning. A tree-structured adaptive neural network for character recognition is proposed here. This architecture has the advantages of structural simplicity, fast learning, and of successive approximation.

*This work is supported in part by a Concordia University Faculty Research Development Program (FRDP) grant. The authors of Telecom Australia Research Labs would also like to acknowledge the permission to publish this paper by the Executive Manager of Telecom Australia Research Labs

2 The Architecture of Neural Tree

The neural tree is essentially a multi-level competitive neural network [8, 13, 22] in which the nodes are organized in the tree topology. A neural tree may have several levels. It is a multi-level, hierarchically organized architecture in which a single winner is selected on each level. The neural tree adopts gradient descent learning to adjust its connection weights.

Typically, the inputs of the nodes of a neural tree are connected to some common inputs $x_0, x_1, ..., x_{n-1}$. Each node n_j computes an error value between the inputs and its link weights $|\mathbf{X}(t) - \mathbf{W}(t)|$ at time t. To perform search on a neural tree, one begins from the root. At any level i, the node with the minimum error value is selected as the winner through competition. The search normally continues along the subtree rooted at the winner node.

Our self-organizing neural tree is different from the tree structured neural network for function approximation [24] in which a function is decomposed first along one dimension of the function to form one level of the tree, and then decomposed along other dimensions to form other levels of the tree. The nodes in the function approximation tree do not represent clusters and that tree does not have the *successive approximation property.*

3 Adaptive Learning for Neural Trees

The neural tree of fixed structure [5] can be easily made adaptable to changing environments by allowing the connection weights to change during application. In the adaptive learning algorithm developed below, a neural tree begins with an empty structure. Nodes are added to the tree when the error rate exceeds a threshold and some nodes are deleted if they remain inactive for a long period.

Since the neural tree is used in hierarchical classification, the learning algorithm should reflect this aspect. The members of a class should have similar

features. Hence we must design an adaptive learning algorithm which meets the criteria: 1) The selection of a winner should take advantage of the successive approximation property; 2) The weights of the winner neuron should be updated to minimize the error.

We need to point out that the adaptive neural tree is a general methodology for classification and vector quantization. Any reasonable measure can be adopted by the adaptive neural tree. The following algorithm employs an error tolerance distance to control the creation of new nodes and a threshold factor to control the splitting of nodes into more nodes [16]. An operational measure is used to control the deletion of nodes from the tree. Let γ_0 be the initial error tolerance distance and θ_0 be the initial error threshold value. The error tolerance and the error threshold change with the tree level. They decrease exponentially with the depth of the tree. We denote the error tolerance and the error threshold at the current level by γ and θ, respectively. Specifically, let $0 < a < 1$ and $0 < b < 1$ be two constants, the error tolerance and threshold for nodes on level k are defined as

$$\gamma = \gamma_0 - a * k \qquad \theta = \theta_0 - b * k \tag{1}$$

The adaptive neural tree algorithm is presented below.

Algorithm Adaptive Neural Tree:

Step 1 : (Prepare input).
Get a new pattern $\vec{x}$
Step 2 : (Initialize variables).
winner $\leftarrow$ root;
$\gamma \leftarrow \gamma_0$;
$\theta \leftarrow \theta_0$;
Step 3 : (Search for a winner node).
WHILE (winner$\neq$leaf & some node meets error tolerance test) DO
$\gamma \leftarrow \gamma_0 - a * k$;
$\theta \leftarrow \theta_0 - b * k$;
min_error $\leftarrow$ inf;
FOR (each child n of winner) DO
n.val $\leftarrow ||\vec{x} - n.\vec{w}||$;
IF (n.val$\leq \gamma$) THEN
IF (n.val < min_error) THEN
keep a record of this node;
END-FOR
IF (no child meets error tolerance test) THEN
IF (there is another slot available) THEN
create a new node;
copy the pattern to the node;
winner $\leftarrow$ new node;
ELSE
winner $\leftarrow$ minimum error child;
ELSE
winner $\leftarrow$ minimum error child that satisfies the error tolerance test;
update(winner, $\vec{x}$);
END-WHILE
Update the operational measure, the accumulated error of the winner, and other parameters;
Step 4 : (Split, delete and collect nodes).
IF (winner.err > threshold) THEN
split-node(winner);
IF (winner.mea < active-threshold)
THEN delete-node(winner);
IF (time to perform node collection)
THEN collect-node(tree);
Go to step 1.

Periodic traversals of the adaptive tree is performed in step 4 to delete those inactive nodes. This step is needed because some inactive nodes may never get referenced. This algorithm has been implemented to execute in three modes, 1) the *grow-only* mode in which no deletion and node collection are performed; 2) the *grow-and-delete* mode in which both node growing and deletion are allowed but the periodic collection of nodes is not allowed, and 3) the *fully adaptive* mode in which growing, deletion and periodic collection are all incorporated.

Let x_l^i denote the lth input element of the ith pattern and $n.w_l$ denote the lth connection weight of neuron n. The squared error,

$$error(n,i) = \sum_{l=0}^{n-1} [x_l^i - n.w_l]^2 \tag{2}$$

is typically used in step 3 to compute the error occurred at a node. This measure is used in the example presented in the paper and in the character recognition experiments. One may choose to use other measures. The following rule is used to modify the weights of a winner root node.

$$n.w_l(t+1) = n.w_l(t) + \alpha \times [x_l^i - n.w_l(t)] \tag{3}$$

where α is the *learning rate* which controls the speed and convergence of learning. In the adaptive tree learning algorithm, the learning rate changes with time exponentially according to the following formula.

$$\alpha(t) = A \times e^{-\beta t} \tag{4}$$

where A is the amplitude and β is the time constant.

The accumulated error for a node n is denoted by $n.err$ and is defined over a time window of finite size. It is calculated each time a node is selected as the winner for a pattern by using the following formula.

$$n.err(t) = n.val(t) + \frac{n.err(t_l)(\omega - t + t_l)\delta[\omega - t + t_l]}{\omega} \tag{5}$$

where δ is the step function, ω is the time window size, and the last time node n was selected as winner was at time t_l. This formula for accumulating errors over time has the effect that the previous accumulated error will be completely lost if the time interval $t - t_l$ is longer than the window size. Otherwise the previous error is measured proportional to $(t - t_l)/\omega$. The operational measure for the activity of node n is defined as follows.

$$n.mea(t) = 1 + n.mea(t_l)[1 - (t - t_l)/\omega]\delta[\omega - t + t_l] \tag{6}$$

When a node n_i is split into two nodes n' and n'', the weights of n' and n'' are copied from n_i with a

random noise added. In vector notation, the weights of the new nodes are defined by

$$n^t.\vec{w} = n.\vec{w} + \vec{r} \tag{7}$$

where t is $'$ or $''$ and $\vec{r}$ is a random noise with zero mean and variance $C\gamma$ for $0 < C < 1$.

The main objective of adaptive neural trees is to attain memory and computation efficiency for changing environments. We want to keep the tree structure small and yet still be able to reduce the error. Hence the tree structure changes with the signal source. It may also discard previous learned patterns once they become inactive after a long period.

The error tolerance distance controls the error bound and the error threshold controls the error rate of a node. Since the upper level nodes represent small a number of classes with a large number of patterns included in each class, they need larger error bounds and larger error rates than the lower level nodes. This is the reason for reducing the error tolerance and the error threshold value in each level further down.

4 Analysis of Adaptive Neural Trees

Theorem. The average error e_n of a node n in a time window frame is bounded above by $max\left(\gamma, \frac{\theta}{\omega} + \frac{\gamma}{\omega}(\gamma - \frac{\theta}{\omega})log_\sigma\left(1 - \frac{\theta}{\gamma\omega}\right)\right)$, where k is the level of the node, ω is the time-window size, $\gamma = \gamma_0 a^k$ is the error tolerance, $\sigma = (\omega - 1)/\omega$ and $\theta = \theta_0 b^k$ is the error threshold value.

Proof. The worst-case scenario occurs when a node has been selected as the winner in the first m consecutive time steps of a time-window frame with an error γ in each step, such that its accumulated error becomes θ at step m. The node is also selected as winner for the remaining steps. After the first m steps, the node is only allowed to have error value $n.val = \theta/\omega$ in each remaining step. We can verify that this is indeed the case by treating it as an optimal greedy heuristic.

For the first m steps, the total accumulated error should be $m\gamma$ in the worst-case. To estimate m, we derive the following relation from equation (5).

$$\theta = \sum_{i=0}^{m-1} \gamma \left(\frac{\omega - 1}{\omega}\right)^i$$

This can be obtained by repeatedly applying equation (5). From this relation we derive $(1 - \sigma)\theta = \gamma(1 - \sigma^m)$which leads to a solution of m

$$m = log_\sigma\left(1 - \frac{\theta}{\gamma\omega}\right) \tag{8}$$

Hence the accumulated error in the first m steps is

$$m\gamma = \gamma log_\sigma\left(1 - \frac{\theta}{\gamma\omega}\right)$$

In the remaining $\omega - m$ steps, the accumulated error is $(\omega - m)\theta/\omega$. Therefore the total accumulated error in this window frame is

$$m\gamma + \frac{\theta}{\omega}(\omega - m) = \theta + \gamma\left(\gamma - \frac{\theta}{\omega}\right) log_\sigma\left(1 - \frac{\theta}{\gamma\omega}\right)$$

The average error is thus bounded by

$$\frac{\theta}{\omega} + \frac{\gamma}{\omega}\left(\gamma - \frac{\theta}{\omega}\right) log_\sigma\left(1 - \frac{\theta}{\gamma\omega}\right) \tag{9}$$

Q.E.D.

This theorem can be used in choosing the parameters for adaptive neural trees to meet the given error requirements. Since it gives only an upper bound which is more conservative than the average error in a stable adaptive tree, the error requirement can be made larger in practice. We are currently investigating equilibrium conditions for the adaptive trees under various distribution functions. Preliminary results will be reported soon.

5 Experimental Study

Example 1. This example demonstrates the behavior of the learning algorithms for training data with three clusters of uniform distribution. The three clusters are generated using independent random variables with uniform distributions. The regions of the three clusters are clearly seen in Figure 1(a)-(f). The 100 patterns of the first cluster, shown as the block of dots on the top of Figure 1, are presented in the first 100 steps. The 400 patterns of the second cluster, shown as the block of dots in the middle of Figure 1, are presented in the next 400 steps. The third cluster has 200 patterns which are shown as the bottom block of dots. The structure of the tree changes when the input data change. The structural change of the adaptive neural tree in different time periods can be clearly observed in Figure 1. Since the patterns are 2D vectors, the plots in Figure 1 have two axes, each representing one component of a vector.

For this example, the initial error threshold is 200 and its change rate $b = 1$. The initial error tolerance γ_0 is 20 and the error tolerance change $a = 0.7$. The learning rate $\alpha = 0.05$ and it decreases by 5% in every 100 time units. The time window size ω is 90 and the periodic node collection is done every 50 time units.

The training samples are shown as small dots, the root node is shown as a filled square, and the tree nodes are shown as small circles. The *parent-child* relationships are indicated by solid lines. The movement of the root node and the change of the tree structure can be seen in the figure. This demonstrates the parametric and structural adaptation of the neural tree.

The statistics collected during each stage are summarized in Table 1. The first column indicates the time step at which the statistical data is collected. The second column gives the average error from step 1 up to the current step. The number of nodes in the tree is given in column 3. Column 4 contains the number of nodes created in step 3 of the algorithm and column 5 contains the number of nodes split in step 4 of the algorithm, during this period of time. Column 5 contains the number of nodes deleted and collected

Time	ave error	tree size	add	split	delete
50	12.93	10	9	3	5
100	10.17	13	8	2	9
150	9.35	18	4	2	3
250	8.74	23	3	3	4
450	8.45	18	2	1	9
700	8.76	21	6	0	3

Table 1: Statistics for example 1

in step 4 of the algorithm. From the statistical figures, we can see that the adaptive neural tree generates new nodes quickly when a new cluster of patterns is presented. The error also reduces quickly and the average error is quite small.

6 Character Pattern Recognition with Neural Trees

We present the results of experimental study on the recognition of a large set characters in this section. Three sets of Chinese characters have been used in our experimental study. One set contains 200 distinct characters, the second set contains 1000 distinct characters and the third set contains 3000 distinct characters.

6.1 Feature Extraction

The transformation-ring-projection (TRP) method [25] which possesses both size and orientation invariant characteristics has been used to extract features for this study. In the TRP, the image transformation technique is employed to center the image and normalize its size; the Ring-Projection scheme is used to handle the orientation problem. Comparing with other research [10, 11, 26], the TRP algorithm has the major merit of being more stable, and the basic operations have been carefully designed so that the proposed algorithm is very simple and regular. The TRP algorithm is briefly described in the following:

Step 1 : Find the center of gravity (X_c, Y_c). This can be computed by the following formulas, and translate it to the origin of the image plane;

$$X_c = \frac{m_{10}}{m_{00}}, Y_c = \frac{m_{01}}{m_{00}},$$

$$m_{\alpha\beta} = \sum_{x=1}^{N}\sum_{y=1}^{N} x^\alpha y^\beta f(x,y),$$

where $f(x,y)$ denotes a pixel of the pattern of size $N \times N$.

Step 2 : Normalizing the size of the input pattern. Let D be the standard size, then the size d_j of the input pattern is normalized by a factor of D/d_j.

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} D/d_j & 0 \\ 0 & D/d_j \end{bmatrix} \begin{bmatrix} x_j \\ y_j \end{bmatrix}$$

Char set	tree size	correct rate	train time
200 chars	331	100%	2 mins
1000 chars	1846	100%	5 mins
3000 chars	10601	100%	15 mins

Table 2: Results for original pattern recognition

Step 3 : Rotation-invariant transformation with Ring-Projection. Projecting the black pixels of the pattern $f^*(i, \theta)$ onto annular multi-layers gives a projection value p_i. In order to stabilize the algorithm, we take

$$P_i = \sum_{i=0}^{i} p_i = \sum_{i=0}^{i}\sum_{\theta=0}^{2\pi} f^*(i,\theta) \quad (i = 0,1,2,...,n$$

Then we can have an n-dimensional vector

$$[P_1 \;\; P_2 \;\; P_3 \;\; ... \;\; P_n]^T$$

or

$$\begin{bmatrix} p_1 \\ p_1 + p_2 \\ p_1 + p_2 + p_3 \\ ..... \\ ..... \\ ..... \\ p_1 + p_2 + ... + p_{n-2} + p_{n-1} \\ p_1 + p_2 + ... + p_{n-1} + p_n \end{bmatrix}.$$

as the feature vector to characterize size-normalized input patterns.

In this method, because P_i, the sum of projections, is used instead of p_i, the individual projection of a ring, the system is more stable than that proposed in [26]. Details of this method can be found in [25].

6.2 Pattern Recognition with Adaptive Neural Tree

The characters are transformed into feature vectors using the TRP method as outlined previously. These feature vectors are then presented to the neural tree algorithm. We have also tested it with various degrees of noise corruption. In the first experiment, each of the three sets of characters is used to train an adaptive neural tree. All the characters in a set are presented to the neural tree for training. The trained adaptive neural trees can separate all the patterns in a set. The results are summarized in Table 2. It is worth mentioning that initial training was performed on multiuser Spark workstations and the training time in the table is response time, not CPU time. The CPU time is significantly shorter.

In the other experiment, we use the original characters for training and noise corrupted characters for tests. In this experiment, we use 2% noise and 5% noise to produce corrupted patterns. These corrupted samples are directly presented to the adaptive neural tree without any filtering operation. Due to time constraint, we have only finished tests on the size 200 and

Char set	tree size	2% noise	5% noise
200 chars	331	93%	91%
1000 chars	1846	95%	93%

Table 3: Results for error corrupted patterns

size 1000 character sets. The results are summarized in Table 3.

The results for the experiments on noise corrupted data indicate that the Ring-Projection transformation produces stable and highly discriminant features. The advantages of the transformation technique and the adaptive neural tree are clearly manifested in these experiments.

7 Comparing with Other Techniques

When compared with other techniques, there is a major advantage of neural networks over the traditional techniques. Neural networks are inherently *adaptive* learning machines. Stable learning neural networks can adapt to complex and changing environments while traditional classification techniques cannot be readily made adaptive. Both *parameter adaptation* and *structural adaptation* can be easily employed in neural networks. Traditional tree classifiers [2, 15, 23] can only be trained off-line.

Hierarchical classification techniques can be either agglomerative or divisive. An agglomerative algorithm is essentially a bottom-up procedure which combines individual samples into groups and small groups into larger groups. Various similarity/dissimilarity measures can be adopted in agglomerative algorithms. Details of the technique and various similarity measures can be found in [15, 19, 23].

A divisive algorithm begins with the entire set of samples. It divides the samples into some groups based on some criteria. These groups can be divided into smaller groups. This process continues until some standards are satisfied or until the leaf nodes contain only individual samples. The division can be based on individual attributes (such is the case for decision trees [21]) or on some similarity/dissimilarity measures.

The tree-structured vector quantizer and the Classification And Regression Tree (CART) are two widely used divisive methods. In vector quantization [7, 20], a tree structured codebook can be constructed for the encoding of signals. A tree-structured vector quantizer is constructed using a sophisticated iterative method. A CART [2, 3, 6] is essentially a decision tree constructed with divisive methods. CART uses supervised learning.

The neural tree learning algorithms differ from the methods discussed in this section in the following ways.

1. Hebbian learning is employed in adaptive neural trees both during training and during execution to achieve parametric adaptation while other methods employ combinatorial or heuristic search algorithms during training and no change is allowed during execution.
2. Structural adaptation is also employed to keep the structure of the tree small so that the search process can be sped up while other methods do not permit structural adaptation.
3. Competition is used to select the winner of each level.

An adaptive neural tree requires a much shorter training time than the frequency sensitive competitive (FSCL) networks [14] and the Kohonen topological map [12]. On a 15MIPs workstation, the training time for a neural tree with 512 leaf nodes and 15480 patterns is less than 15 minutes while the training time for FSCL networks and Kohonen networks with 512 nodes require more than two hours. For larger size networks, the difference between training times is even larger. It is important to note that the neural tree has the *successive approximation* property which is essential for quick search of the tree.

8 Summary

An adaptive neural tree architecture and its application to the recognition of a large set character patterns have been studied in this paper. This neural architecture is capable of performing hierarchical classification of patterns. In our experiment of large set character recognition, the neural trees have demonstrated impressive performance and the preliminary results are very encouraging. The neural tree can be quickly generated and its speed is significantly higher than tradition techniques and other neural network techniques [12, 14, 16, 20]. The large set character pattern experiment shows the practicality of the neural tree architecture.

References

[1] E. Alpaydin (1990), "Grow and learn: an incremental method for category learning", *Internat. Neural Network Conf.*, Paris, France 1990.

[2] L. Breiman, J.H Friedman, R.A. Olshen and C.J. Stone (1984), *Classification And Regression Trees*, Wadsworth and Brooks, Monterey CA.

[3] P.A. Chou (1991), "Optimal partitioning for classification and regression trees", *IEEE Trans. PAMI*, Vol.13, No.4, pp.340-354, 1991.

[4] S.E. Fahlman and C. Lebiere (1990), "The cascade-correlation architecture", in *Advances in Neural Information Processing Systems*, Vol.2, (D.S. Touretzky, Ed.), pp524-532.

[5] L-Y. Fang, et al, (1991), "Hierarchical classification with neural trees", proc. *IJCNN 1991*, Singapore, Nov. 1991.

[6] S.B. Gelfand, C.S. Ravishankar, and E.J. Delp (1991), "An iterative growing and pruning algorithm for classification tree design", *IEEE Trans. PAMI*, Vol.13, No.2, pp.163-174, 1991.

[7] R.M. Gray (1984), "Vector quantization", *IEEE ASSP Mag.*, Vol.1, No.2, pp.4-29, 1984.

[8] S. Grossberg (1987), "Competitive learning: from interactive activation to adaptive resonance", *Cognitive Science*, Vol.11, pp.23-61, 1987.

[9] G.E. Hinton (1989), "Connectionist learning procedures", *Artificial Intelligence*, Vol.40, No.1-3., pp.185-234, 1989.

[10] Yuan-Neng Hsu, H. H. Arsenault and G. April (1982), "Rotation-invariant digital pattern recognition using circular harmonic expansion," *Applied Optics*, Vol.21, No.22, pp.4012-4015, Nov. 1982.

[11] S. Kahan, T. Pavlidis and H. S. Baird, "On the recognition of printed characters of any font and size," *IEEE Trans. Pattern Anal. and Machine Intelligence*, Vol. 9, No. 2, pp. 274-288, March 1987.

[12] T. Kohonen (1988), *Self-organization and Associative Memory*, 2nd Ed. Springer-Verlag, New York, 1988.

[13] B. Kosko (1991), "Stochastic competitive learning", *IEEE Trans. Neural Networks*, Vol2. No.5, pp.522-529, September 1991.

[14] A.K. Krishnamurthy, S.C. Ahalt, D.E. Melton, and P. Chen (1990), "Neural networks for vector quantization of speech and images", *IEEE Journal of Selected Areas in Communications*, Vol.8, No.8, pp.1449-1457, Oct. 1990.

[15] G.N. Lance and W.T. Williams (1968), "Mixed-data classifactory programs II. Divisive systems", Aust. Computer Journal, **1**, pp.82-85.

[16] T-C. Lee and A.M. Peterson (1990), "Adaptive vector quantization using a self-development neural networks", *IEEE Journal of Selected Areas in Communications*, Vol.8, No.8, i pp.1458-1471, Oct. 1990.

[17] T. Li, et at, (1991), "Hierarchical classification and vector quantization with neural trees", *Technical Report*, Department of Computer Science, Concordia University, Montreal Canada.

[18] R.R. Lippmann (1987), "An introduction to computing with neural nets", *IEEE ASSP Mag.*, Vol. 4, No. 2, pp.4-22, 1987.

[19] P. McNauton-Smith, W.T. Williams, M.B. Dale, and L.G. Mockett (1964), "Dissimilarity analysis: a new technique of hierarchical subdivision", *Nature*, **202**, pp.1034-11035.

[20] N.M. Nasrabadi and Y. Feng (1988), "Vector quantization of images based upon the Kohonen self-organizing feature maps", *Proc. IEEE Int. Conf. Neural Networks*, pp.1101-1108, 1988.

[21] J.R. Quinlan (1985), "Decision trees and multi-valued attributes", in *Machine Learning*, Vol.11, (J.E. Hayes and D. Michie, Eds), Oxford University Press.

[22] D.E. Rumelhart and D. Zipser (1985), "Feature discovery by competitive learning", *Cognitive Science*, **9**, pp.75-112.

[23] R.R. Sokal and P.H.A. Sneath (1963), *Numerical Taxonomy*, W.H. Freeman, San Francisco and London, 1963.

[24] T.D. Sanger, "A tree-structured adaptive network for function approximation in high-dimension spaces", *IEEE Trans. Neural Networks*, Vol.2, No.2, pp.285-293, 1991.

[25] Y.Y. Tang, H.D. Cheng and C.Y. Suen (1991), "Transformation-ring -projection (TRP) algorithm and its VLSI implementation", *Internat. Journal of Pattern Recognition and Artificial Intelligence*, Vol.5, No.1-2, pp25-56.

[26] A. Taza and Ching Y. Suen, "Discrimination of planar shapes using shape matrices," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol.19, No.5, pp.1281-1289, Sept. 1989.

[27] Q-R. Wang and C.Y. Suen (1984), "Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition", *IEEE Trans. PAMI*, Vol.6, No.4, pp406-417.

[28] L. Xu (1990), "Adding learned expectation into the learning procedure of self-organizing maps", *Int. Journal of Neural Systems*, Vol.1, No.3, pp.269-284.

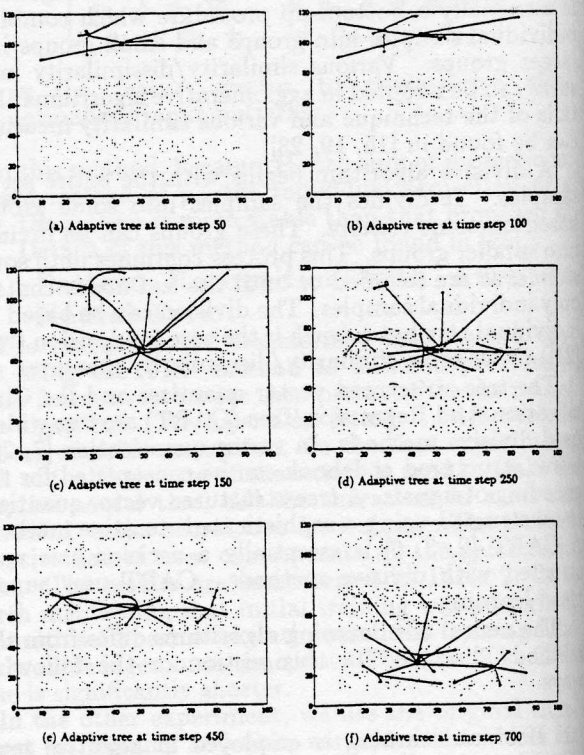

Figure 1: Training Set with Three Uniform Clusters