GENERATING-SHRINKING ALGORITHM FOR LEARNING ARBITRARY CLASSIFICATION

Yan Qiu CHEN, David W. THOMAS, and Mark S. NIXON

Department of Electronics and Computer Science

University of Southampton, U.K.

GENERATING-SHRINKING ALGORITHM FOR LEARNING ARBITRARY CLASSIFICATION

Abstract

This paper proposes a novel generating-shrinking algorithm, which builds and then shrinks a three-layer feed-forward neural network to achieve arbitrary classification in n-dimensional Euclidean space. The algorithm offers guaranteed convergence to a 100% correct classification rate on training patterns. Decision regions resulting from the algorithm are analytically described, so the generalisation behaviour of the trained network is analytically known. By altering the value of a reference number, the trained neural classifier can achieve scale-invariant generalisation as well as equal-distance generalisation to accommodate different requirements.

Keywords

Generating-shrinking algorithm, arbitrary classification, three-layer feed-forward networks, generalisation.

List of symbols

$\mathcal{R}^n$	n-dimensional Euclidean space
$\mathcal{R}$	the set of all real numbers
$\mathcal{N}$	the set of all natural numbers
$\|.\|$	the Euclidean norm
$\in$	in
$\notin$	not in
$\subset$	subset of
$\cup$	union
$\cap$	intersection
$\sum$	sum
$\prod$	product
$\forall$	for all
$\Rightarrow$	imply
$\angle$	angle
$\square$	the end of a proof
p	pattern vector
p'	extended pattern vector
r	the reference number
n_T	number of training patterns
n_C	number of classes
M	the mapping associating each training pattern with its true class
Li	the i-th layer
n^{Li}	number of neurons in the i-th layer
o_i^{Lj}	output of the i-th neuron in the j-th layer
w_{ij}^{Lk}	weight from the j-th neuron in the $(k-1)$-th layer to the i-th neuron in the k-th layer
D_i	decision region of the i-th class

1 Introduction

Neural networks used as classifiers have recently been studied intensively [1], [2], [3]. Trained networks are able to classify patterns that appeared at the training stage (training patterns) as well as to generalise what it has learned, namely, to classify patterns that did not appear in training (these patterns shall be referred to as new patterns). The commonly-used training method is to use the back-propagation algorithm [4], which can be applied to binary (Boolean) patterns or real patterns (vectors in n-dimensional Euclidean space).

Back-propagation is essentially a gradient descent search in the weight space to minimise an error function. As with other algorithms employing the gradient descent search strategy, there exists the possibility that the algorithm gets stuck in a local minimum. In other words, it is not guaranteed to converge to the global solution although it reportedly converges in most cases. The algorithm is slow, it can take hours for a moderate-sized network to converge on a fast PC. Back-propagation is sensitive to its initial conditions [5], generalisation of the trained network depends on the choice of a set of initial conditions such as the network size and initial random weights. Over-generalisation is also reported [6]. It is generally difficult to stop training at a point where generalisation is good.

Another learning scheme is to use a Hamming net [7], [8]. The first layer of the net calculates the Hamming distances between the input pattern and exemplars. Since the Hamming distance is defined only over binary patterns, the input pattern to the Hamming net, as a result, is restricted to binary patterns. The second-layer of the net needs iterations; The number of steps required is of the order of the number of neurons for the problem.

Recently, Mezard et al. [9] proposed a tiling algorithm for building a multi-layer feed-forward network to classify Boolean patterns with guaranteed convergence. Zollner et al. [10] offered another algorithm for building a three-layer feed-forward network to classify Boolean patterns with guaranteed convergence. Both studies conclude with the possibility of extending their methods to real vectors, though no further details are presented. In their studies, generalisation is demonstrated by simulations, though no analytical generalisation rule is given.

In this paper, we propose a novel generating-shrinking algorithm, which builds and then shrinks a three-layer feed-forward neural network to classify arbitrary patterns in n-dimensional Euclidean space $\mathcal{R}^n$. The algorithm offers a fast guaranteed convergence to a 100% correct classification rate on training patterns. The decision regions formed by the algorithm are analytically described, hence the generalisation behaviour is analytically known. By altering the value of a reference number, one can obtain scale-invariant generalisation as well as equal-distance generalisation.

At the time of training, one is given a finite set of training patterns $\{p_1, p_2, ..., p_{n_T}\}$, where n_T is the number of training patterns. There is a finite set of classes $\{C_1, C_2, ..., C_{n_C}\}$, where n_C is the number of classes. Each training pattern belongs to one of the classes, and a mapping M can be defined such that for each training pattern p, $M(p)$ is the ordinal number of its class. e.g., if p_i belongs to C_j then $M(p_i) = j$.

There are two possibilities for a pattern to be classified: one is that the pattern is one of the training patterns; the other is that the pattern is a new pattern. If a training pattern p is currently the input to a trained network, the network output is required to be equal to $M(p)$, which means that the network functions like a lookup table and classifies the pattern p into the $M(p)$-th class. If a new pattern is applied, then the classification

should be governed by some generalisation mechanism that enables the trained network to choose a class best fitting the input pattern. The subset of $\mathcal{R}^n$ where patterns are classified into the m-th class is termed the decision region of the m-th class.

The correct classification rate is defined to be

$$CCR = \frac{n_{correct}}{n_{total}} \times 100\ (\%), \tag{1}$$

where $n_{correct}$ is the number of patterns correctly classified, n_{total} is the total number of patterns.

2 Architecture

A three-layer feed-forward architecture is used. The first layer (input layer, denoted by $L1$) comprises linear neurons, the second layer (hidden layer, denoted by $L2$) sigma-pi neurons and the third layer (output layer, denoted by $L3$) linear threshold neurons. The number of neurons in the first and second layers is dynamically determined by the algorithm while the number of neurons in the third layer equals the number of classes.

The first layer of the network receives an n-dimensional pattern vector $p \in \mathcal{R}^n$ to be classified, along with a fixed reference number $r \in \mathcal{R}$. Together, they form an $(n+1)$-dimensional vector $(p, r) \in \mathcal{R}^{n+1}$ which shall be later referred to as the extended input pattern and denoted by p'. The outputs of the neurons in the third layer form a binary n-tuple $(o_1^{L3}, o_2^{L3}, ..., o_{n_C}^{L3})$. The i-th component in this n-tuple is associated with the i-th class, that is to say $o_i^{L3} = 1$ means the input pattern falls into the i-th class. In this sense,

the network output is defined to be the ordinal number of the neuron in the output layer whose output is 1.

The input-output relations of each layer are described respectively by the following three equations.

$$o_i^{L1} = \sum_{j=1}^{n+1} w_{ij}^{L1} p'_j \quad \text{for } 1 \le i \le n^{L1}, \tag{2}$$

where o_i^{L1} is the output of the i-th neuron in the first layer, w_{ij}^{L1} is the weight of the connection between the j-th input component and the i-th neuron in the first layer, n^{L1} is the number of neurons in the first layer, and p'_j is the j-th component of p'.

$$o_i^{L2} = \prod_{j=1\ (j\neq i)}^{n^{L1}} u(w_{ij}^{L2} o_j^{L1} + w_{ii}^{L2} o_i^{L1}) \quad \text{for } 1 \le i \le n^{L2}, \tag{3}$$

$$\text{where } u(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \le 0 \end{cases},$$

o_i^{L2} is the output of the i-th neuron in the second layer, w_{ij}^{L2} is the weight of the connection between the output of the j-th neuron in the first layer and the i-th neuron in the second layer, and n^{L2} is the number of neurons in the second layer.

$$o_i^{L3} = u(\sum_{j=1}^{n^{L2}} w_{ij}^{L3} o_j^{L2} + \theta_i) \quad \text{for } 1 \le i \le n^{L3}, \tag{4}$$

where $\theta_i = -0.5$ for $1 \le i \le n^{L3}$, o_i^{L3} is the output of the i-th neuron in the third layer,

w_{ij}^{L3} is the weight of the connection between the output of the j-th neuron in the second layer and the i-th neuron in the third layer, and n^{L3} is the number of neurons in the third layer.

3 The Algorithm

The training algorithm consists of two phases. During the first phase, a network is constructed, and then, at the second phase, unnecessary neurons and connections are removed. Figure 1 is a flow chart of the second phase.

3.1 Phase 1

1. Construct a network such that the neurons are chosen as mentioned in section 2 and the number of neurons in each layer is

$$n^{L1} = n^{L2} = n_T, \quad n^{L3} = n_C. \tag{5}$$

2. Choose the reference number r, as will be discussed in section 4.3, and fix it throughout the training and application of the network.

3. For $k = 1, 2, ..., n_T, \quad j = 1, 2, ..., n + 1$, do the following

$$w_{kj}^{L1} = \begin{cases} cp_{k,j} & \text{if } 1 \le j \le n \\ cr & \text{if } j = n+1 \end{cases} \tag{6}$$

where $p_{k,j}$ is the j-th component of the k-th training pattern p_k, $c = \frac{1}{\sqrt{r^2 + \sum_{i=1}^{n} p_{k,i}^2}}$

4. For $i = 1, 2, ..., n_T$, $j = 1, 2, ..., n_T$, do the following

$$w_{ij}^{L2} = \begin{cases} 1 & \text{if } i = j \\ -1 & \text{if } i \ne j \end{cases} \tag{7}$$

5. For $k = 1, 2, ..., n_T$, $i = 1, 2, ..., n_C$, do the following

$$w_{ik}^{L3} = \begin{cases} 1 & \text{if } i = M(p_k) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

3.2 Phase 2

1. Define a logical variable "removed_flag" to indicate whether any neurons have been eliminated in the current pass. Define a dynamical array to contain an index set I of integers with each element being the ordinal number of each neuron in the first layer and in the second layer.

2. Initialise the index set I to be $\{i \in \mathcal{N} : 1 \le i \le n^{L1}\}$, ($n^{L1}$ is the current number of the neurons in the first layer.) and initialise the logical variable "removed_flag" to be "false".

3. Randomly choose an element $i \in I$, remove it from the set I, and temporarily remove

the associated i-th neuron in the first layer and in the second layer.

4. Test the remaining network against all training patterns, if it can classify every pattern correctly then go to step 5, otherwise go to step 6.

5. Permanently remove the temporarily removed neurons, and set the "removed_flag" to be "true". If there is any element in I then go to step 3, otherwise go to step 2.

6. Recover the temporarily removed neurons. If there is any element in I then go to step 3, otherwise go to step 7.

7. If the "removed_flag" is "true" then go to step 2, otherwise the algorithm terminates.

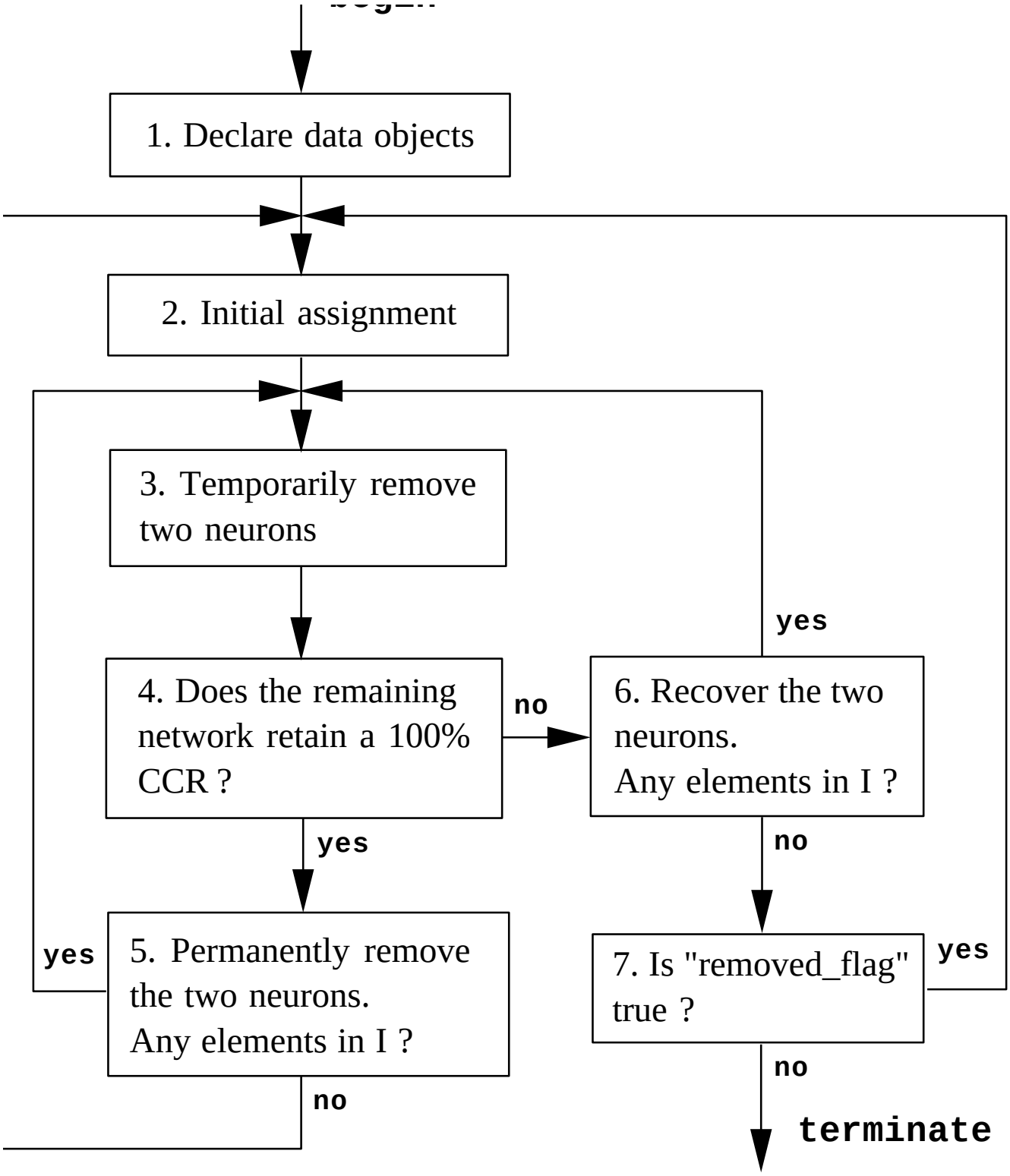


Figure 1: Flow chart of the second phase of the algorithm. (Numbers in the chart refer to the steps in section 3.2 .)

4 Analysis of the Algorithm

This section summarises the analytical results we obtained. For a formal treatment, see the appendices.

4.1 Performance

It is proved that the network constructed in the first phase is guaranteed to achieve a 100% correct classification rate on training patterns (theorem 1). The shrinking process in the second phase is designed to remove surplus neurons after ensuring that the remaining network retains its 100% correct classification rate. By induction, therefore, the shrunk network also achieves a 100% correct classification rate.

The amount of computation required in the first phase is roughly proportional to the number of training patterns and the dimension. It is finite and can be assessed a priori, providing one knows the number and the dimension. The amount of computation needed in the second phase varies with the number of neurons eliminated. However, an upper bound is obtainable by considering the worst case where each and every neuron is eliminated in a single pass. i.e., the number of iterations needed equals the number of neurons removed.

4.2 Decision regions

Consider the case that there are n_T training patterns. For the k-th pattern p_k which is a member of the m-th class, there are $n_T - 1$ hyper-planes with each dividing $\mathcal{R}^n$ into two

regions such that one region contains p_k and the other contains p_i, $i \neq k$. By intersecting all these $n_T - 1$ regions that contain p_k, a region, denoted by H_k, is obtained where a pattern is classified into the m-th class.

For each training pattern p_k , there corresponds a region H_k containing p_k. If there is more than one training pattern belonging to the m-th class, then the decision region of the m-th class is simply the union of all above-mentioned regions H_k for which $M(p_k) = m$.

4.3 Generalisation

As the reference number r approaches zero, all the hyper-planes pass through the origin and separate training patterns in an equal-angle way. As a result, all the decision regions are hyper-polyhedron with apexes located at the origin, hence the classifier is scale-invariant.

Scale-invariant classifiers have many application areas. For many classification problems, the length of the feature vector reflects the "strength" of the observed objects, (e.g. intensity or contrast of an image, volume of a piece of speech) while the direction of the vector characterises the object. In these cases, one wishes to use a scale-invariant classifier to classify objects of the same characteristics with different "strength" into the same class.

As r tends to infinity, the hyper-planes separate training patterns in an equal-distance manner. In other words, the neural classifier trained by the algorithm functions like a nearest-neighbour classifier as r tends to infinity. This shows that a multi-layer feed-forward network can be trained to achieve the well-known nearest neighbour rule. Nearest-neighbour classifiers have been well studied and many application areas have been found

[11].

5 Experimental Results

The generating shrinking algorithm along with the back-propagation method were tested using the two spiral problem and the texture classification problem. All the tests were written in C and run on a 25MHz PC-486.

5.1 The two spiral problem

The two spiral problem has been used for testing neural classifiers [12], [13]. Two data sets, one for training, and the other for testing, were randomly generated. The testing data set was generated using a less concentrated probability density function than the training data (prototype) set to evaluate generalisation capability of a trained network, i.e., the testing data points are more scattered around the two spirals than the training data points; see figure 2 and figure 3.

A three-layer feed-forward network was built and shrunk by the generating shrinking algorithm using the training set. The trained network (the initial network, moderately shrunk network, and the maximally shrunk network) was then tested against the testing set. Presented in table 1 and figure 4 are the results for $r = 500$ (equal-distance classification), which show (1) both the initial un-shrunk network constructed during phase 1 and the network after shrinking achieve a 100% correct classification rate on training patterns, as asserted by theorem 1. (2) the algorithm is very fast: only 0.2 seconds were required to build (train) a network, and 176 seconds were used to shrink the built network. (3)

generalisation performance of the initial trained network is very good, while that of the shrunk network is satisfactory although some deterioration in generalisation performance is observed. (4) the size of the maximally shrunk network is considerably smaller than that of the initial network.

The back-propagation method was also tested using the same data base for comparison. Three-layer perceptrons with various numbers of neurons were used.

Case 1. 10 neurons in the first layer, 10 in the second, and 2 in the third. The classification behaviour of the network at some epochs are shown in figure 5. After initial 3500 epochs, the network reached a solution (a minimum in terms of the error function) and then suddenly escaped from it. After another 7500 epochs it managed to arrive at another solution which is apparently worse than the previous one. It soon escaped again and embarked on the process of trying to find yet another solution.

One might impose some criterion on the error function to stop iterations at an acceptable solution as we manually did at its first and second solutions where the trained network was tested against the testing set. The results are shown in table 2.

Case 2. 20 neurons in the first layer, 20 in the second, and 2 in the third. The network of this size reached a stable solution after 3000 epochs as shown in figure 6. The trained network was tested against the testing set, and the results are shown in table 3.

Some other network sizes were also tried. The non-convergence problem was experienced in some cases. In summary, (1) back-propagation is not guaranteed to converge. (2) it takes a long time - a few hours on a fast PC - to converge. (3) solutions can be local minima where the correct classification rate on training patterns is not guaranteed to be

100%. (4) generalisations depends upon the network size, it can be satisfactory as well as poor, the generalisation behaviour is generally not predictable or controllable. This might be explained as the decision boundaries are formed in a gradient descent search that minimises the error function that only takes into account the network behaviour at the training data points. There is no guide how decisions boundaries are formed between these points although sufficient data may help mitigate this problem.

5.2 The texture classification problem

Three texture pictures in Brodatz's photographic atlas of textures [14]: D4 (pressed cork), D21 (French canvas), and D56 (straw matting) were used. Each picture was scanned by an HP flat bed scanner to produce a $256 \times 256 \times 8$ digital image, from which, sixteen $64 \times 64 \times 8$ sub-images were obtained using perfectly aligned windows. Nine of them were randomly chosen.

Feature vectors were obtained using the 2-dimensional discrete Fourier transform on the sub-images and then applying rings to the coefficients [15]. To visualise the results, feature vectors were chosen to be 2-dimensional. The training data set is shown in figure 7. To test generalisation capability of the trained classifier, a testing set was obtained using the same textures but under various contrast, as shown in figure 8.

The classification results of the generating-shrinking algorithm for $r = 0.01$ (scale-invariant classification) on this problem are presented in figure 9 and table 4, which show (1) the trained network achieves a 100% correct rate on training patterns. (2) the algorithm is very fast. (3) generalisation is very good; the trained network (both the initial and the shrunk) achieves a 100% correct rate on testing patterns. This is because the scale-invariant generalisation is what is required by the texture classification

problem to achieve contrast-invariance which is obviously desirable because the contrast of images of the same texture can vary greatly due to variations of illumination, camera, and digitisation equipments etc. (4) the shrunk network is considerably smaller than the initial network.

The back-propagation method was also tested on this problem. A three-layer perceptron with 10 neurons in the first layer, 10 in the second, and 3 in the third was used. After 2000 epochs, the network reached a stable solution as shown in figure 10. The classification results against the testing data set are shown in table 5, from which we see that (1) the algorithm took 20 minutes to converge which is much longer than that by the generating shrinking algorithm though it is shorter than that in the two spiral problem. (2) generalisation is poor. This may be explained as the decision regions are formed by gradient descent searches for a minimum in the error function that is concerned with network behaviour only on points where there is a training pattern. Generalisation capability is not reflected in the function hence it is not optimised during the searches. Thus, in this problem and in most problems, it is unlikely that the network trained by back-propagation happens to have the desired generalisation - scale-invariance for this problem.

Table 1: Results of the generating-shrinking algorithm on the two spiral problem for $r = 500$. CCR: correct classification rate, shrunk net 1: intermediately shrunk network, shrunk net 2: maximally shrunk network.

	initial net	shrunk net 1	shrunk net 2
number of neurons in 1st layer	500	100	36
number of neurons in 2nd layer	500	100	36
number of neurons in 3rd layer	2	2	2
processing time (seconds)	0.2	123	176
CCR on training patterns (%)	100	100	100
CCR on testing patterns (%)	100	95.8	91.4

Table 2: Results of the back-propagation algorithm on the two spiral problem. CCR: correct classification rate.

	first solution	second solution
number of neurons in 1st layer	10	
number of neurons in 2nd layer	10	
number of neurons in 3rd layer	2	
processing time (seconds)	3780	11590
CCR on training patterns (%)	97.8	91.8
CCR on testing patterns (%)	85.2	77.6

Table 3: Results of the back-propagation algorithm on the two spiral problem. CCR: correct classification rate.

	the solution
number of neurons in 1st layer	20
number of neurons in 2nd layer	20
number of neurons in 3rd layer	2
processing time (seconds)	10180
CCR on training patterns (%)	100
CCR on testing patterns (%)	89.6

Table 4: Results of the generating-shrinking algorithm on the texture classification problem for $r = 0.01$. CCR: correct classification rate.

	initial net	the shrunk net
number of neurons in 1st layer	27	3
number of neurons in 2nd layer	27	3
number of neurons in 3rd layer	3	3
processing time (seconds)	< 0.1	2.2
CCR on training patterns (%)	100	100
CCR on testing patterns (%)	100	100

Table 5: Results of the back-propagation algorithm on the texture classification problem. CCR: correct classification rate.

	the solution
number of neurons in 1st layer	10
number of neurons in 2nd layer	10
number of neurons in 3rd layer	3
processing time (seconds)	1260
CCR on training patterns (%)	100
CCR on testing patterns (%)	67.7

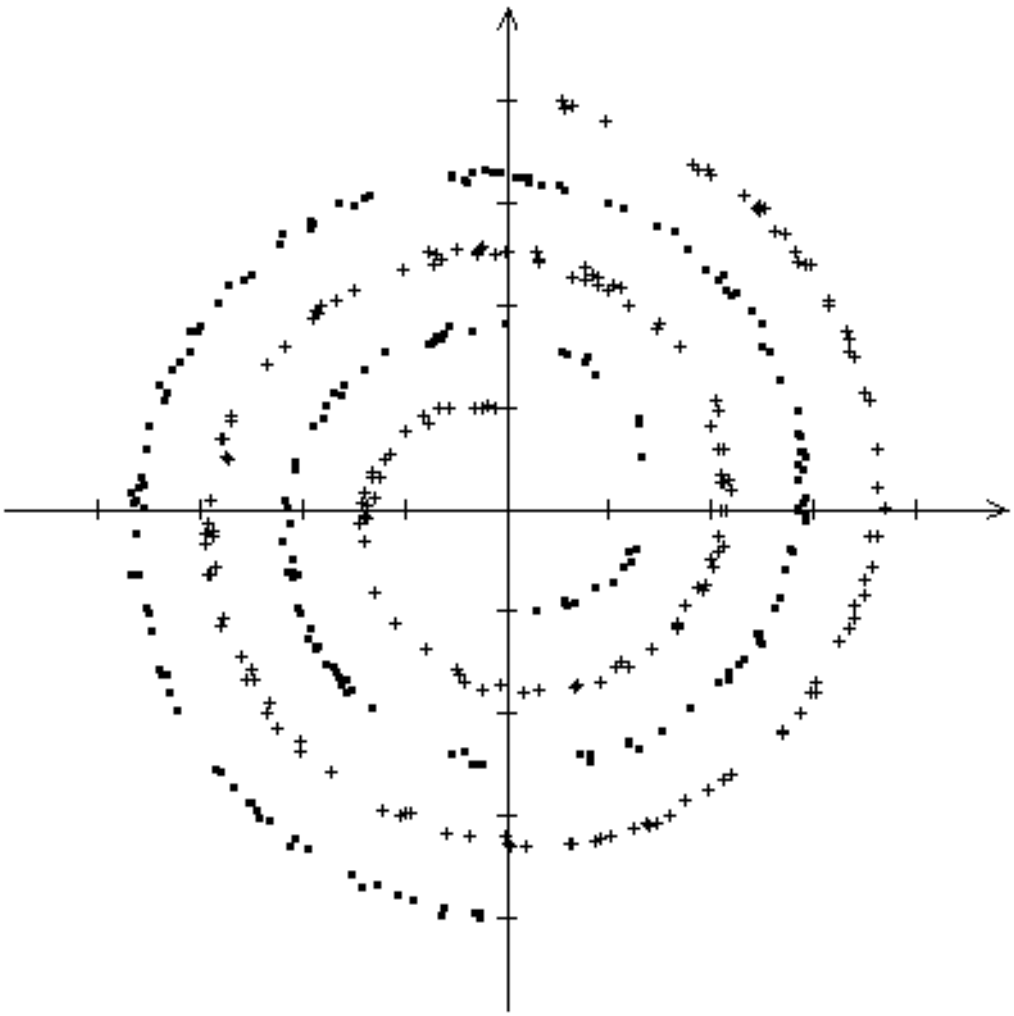


Figure 2: 400 training patterns of the two spiral problem, marked by · and + respectively. Scale: 1 / division.

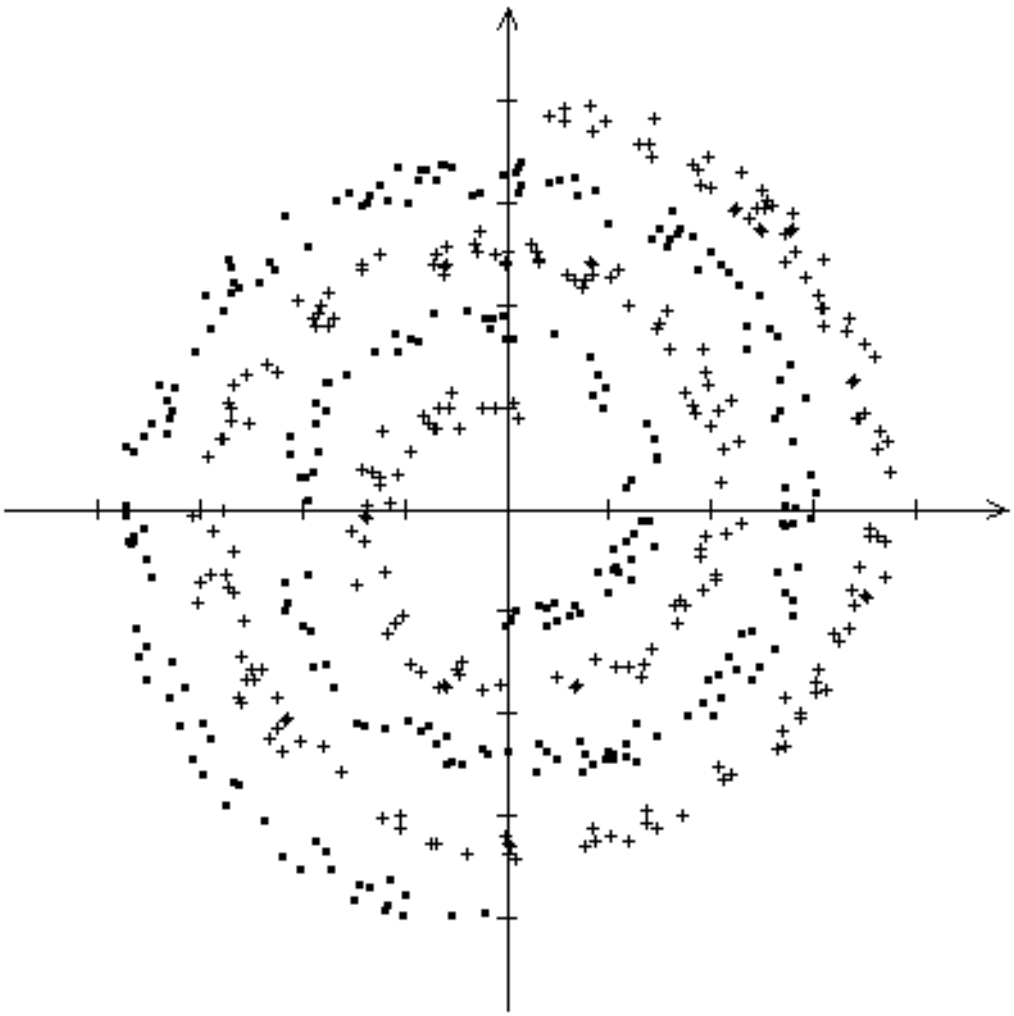


Figure 3: 400 testing patterns of the two spiral problem, marked by · and + respectively. Scale: 1 / division.

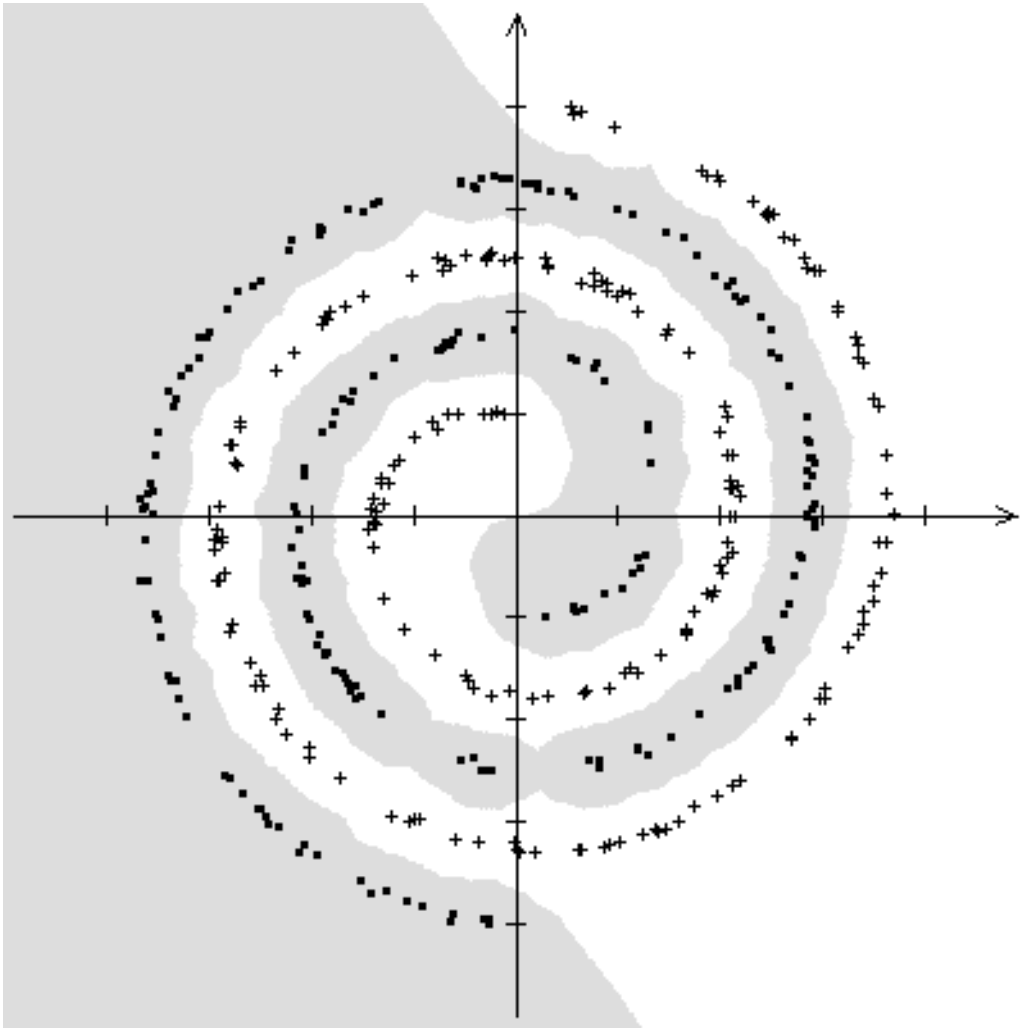
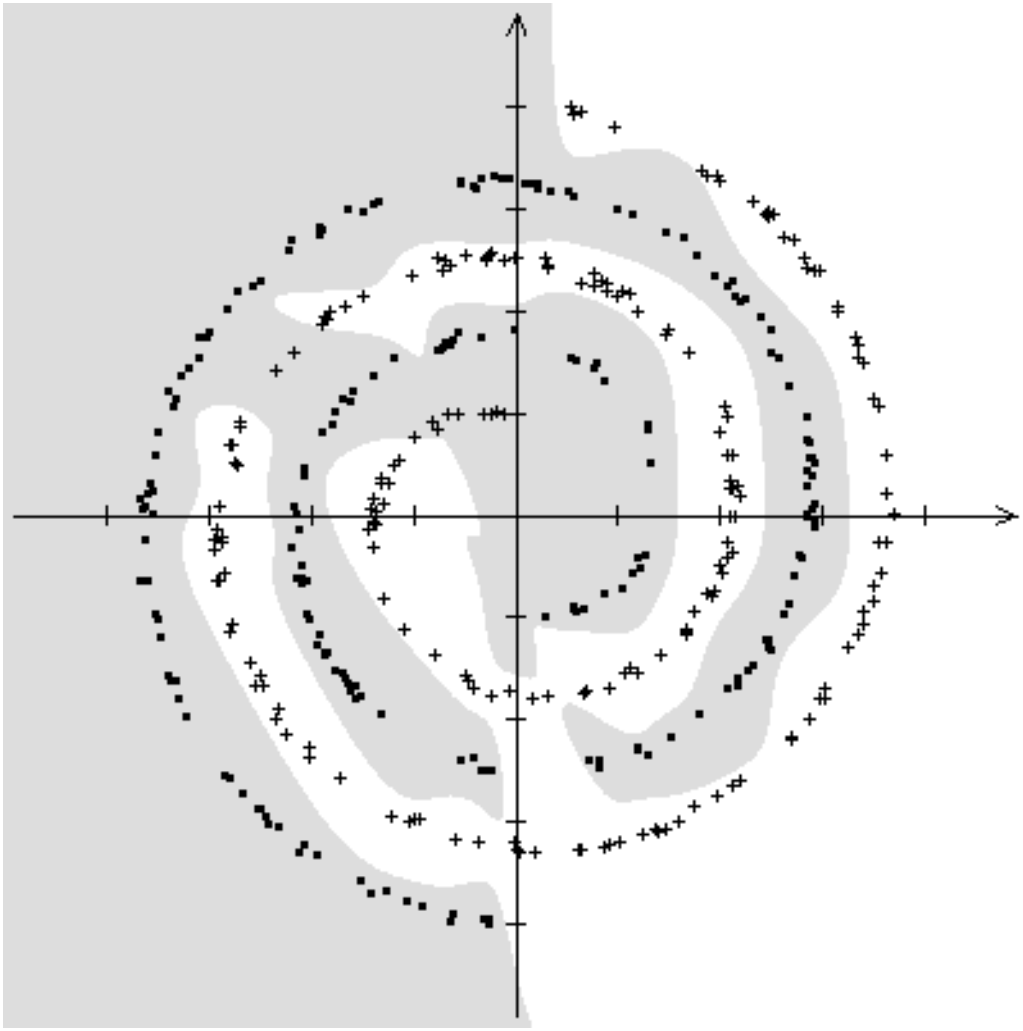
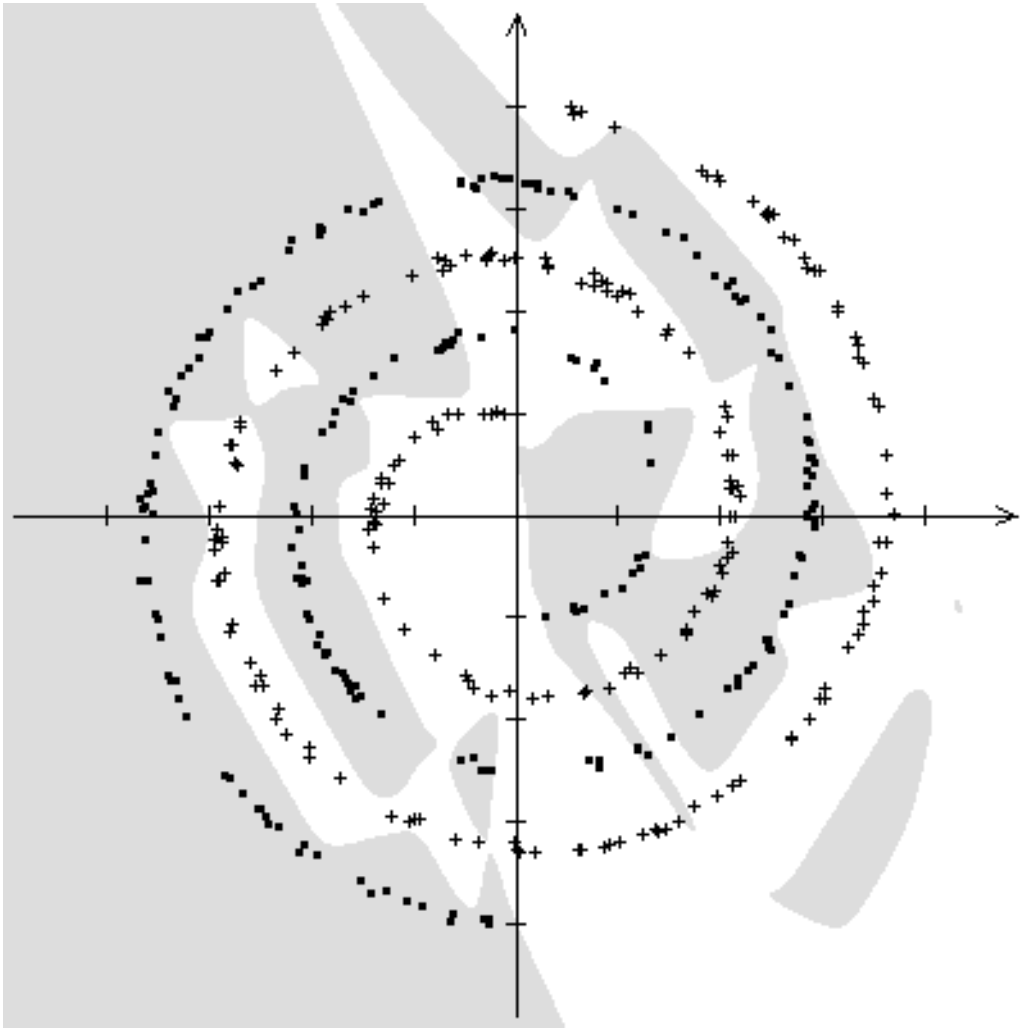


Figure 4: Classification results by the generating shrinking algorithm (equal-distance) with decision regions superimposed on the training patterns.

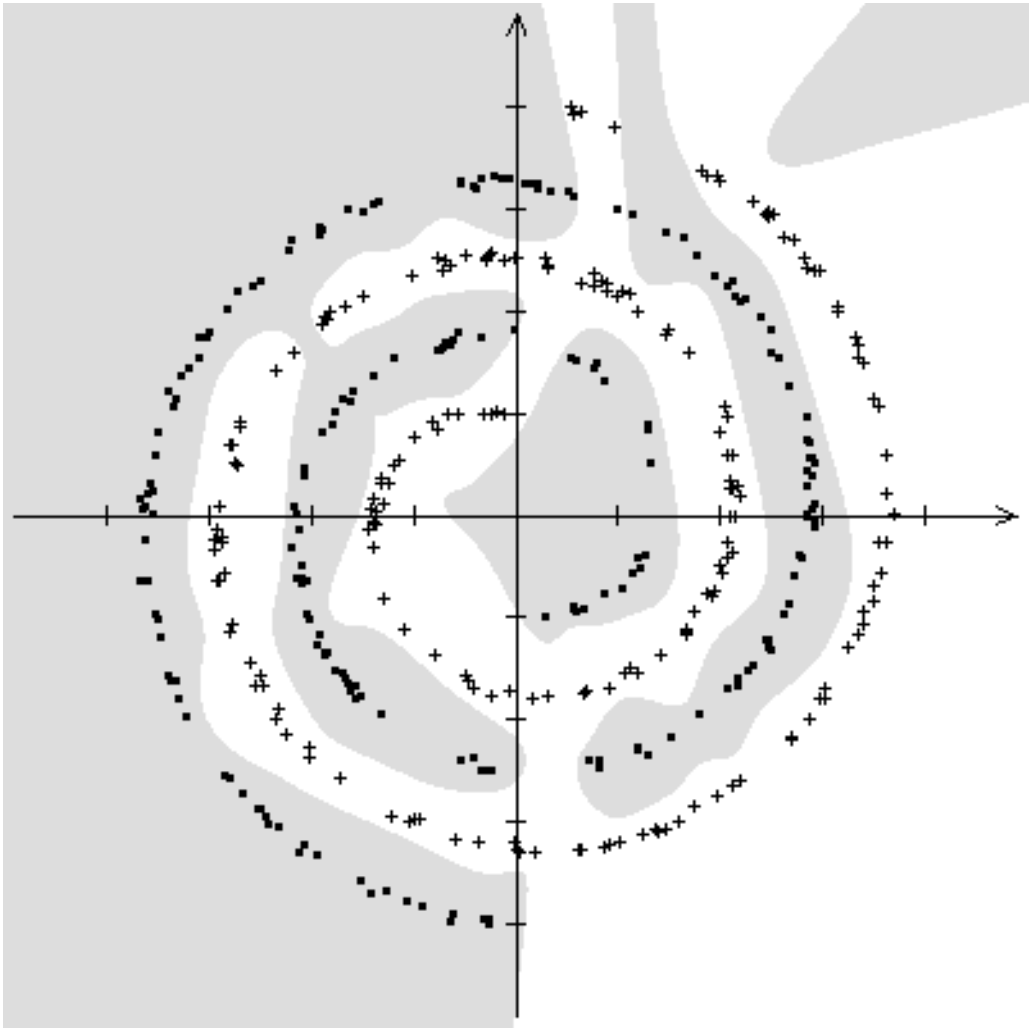


(a) After 3500 epochs

Figure 5: Classification results by the back-propagation method (three-layer perceptron, 10 neurons in the first layer, 10 in the second, and 2 in the third) with decision regions superimposed on the training patterns.



(b) After 10500 epochs



After 3000 epochs

Figure 6: Classification results by the back-propagation method (three-layer perceptron, 20 neurons in the first layer, 20 in the second, and 2 in the third) with decision regions superimposed on the training patterns.

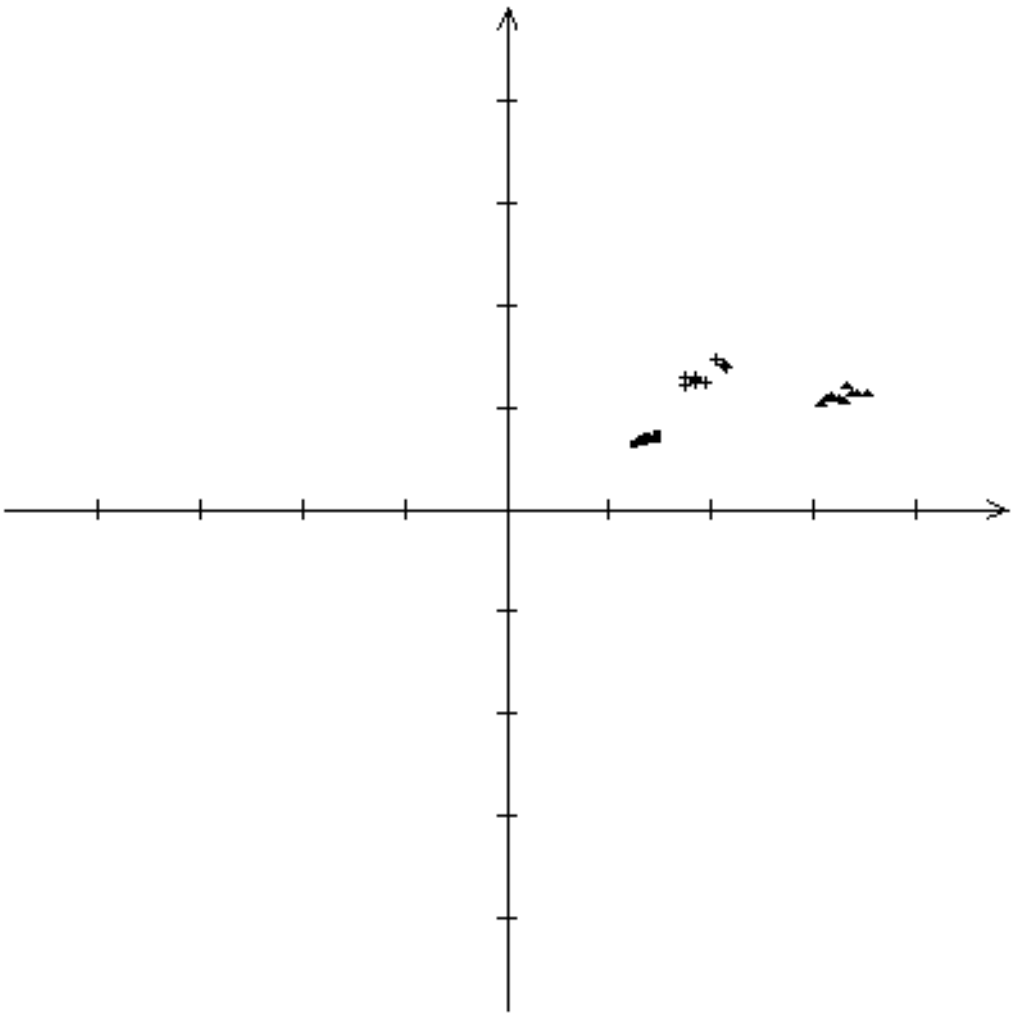


Figure 7: Training patterns from the texture classification problem, D4 (pressed cork) marked by ·, D21 (French canvas) by +, and D56 (straw matting) by △. Scale: 1 / division.

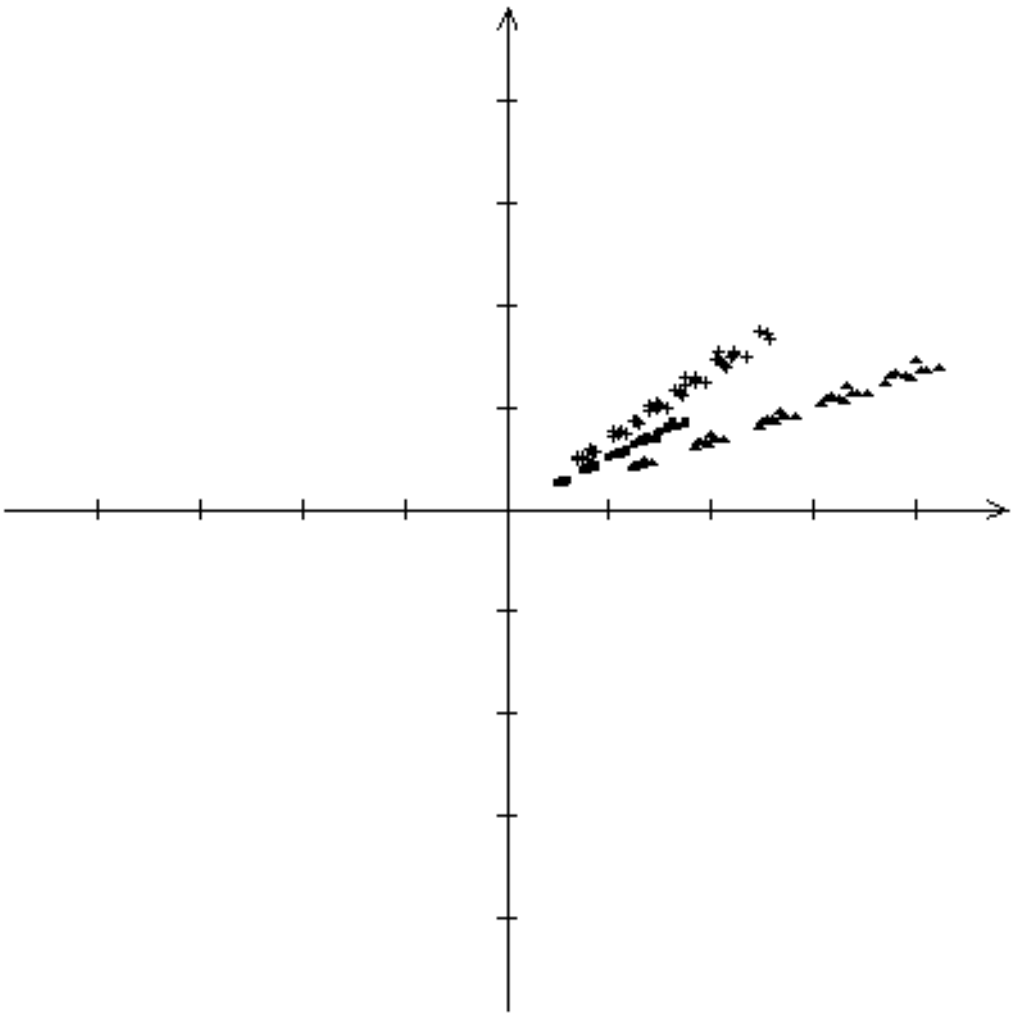


Figure 8: Testing patterns from the texture classification problem, D4 (pressed cork) marked by $\cdot$, D21 (French canvas) by $+$, and D56 (straw matting) by $\triangle$. Scale: 1 / division.

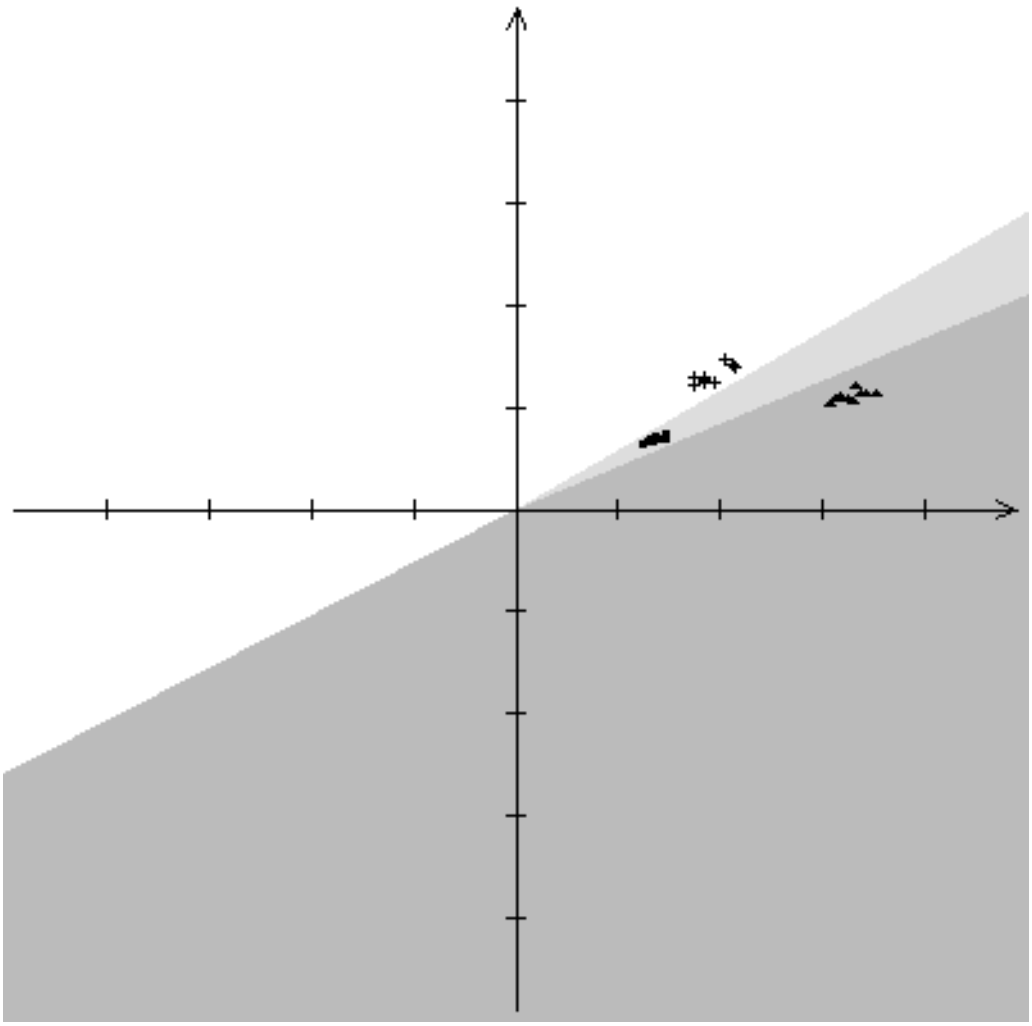
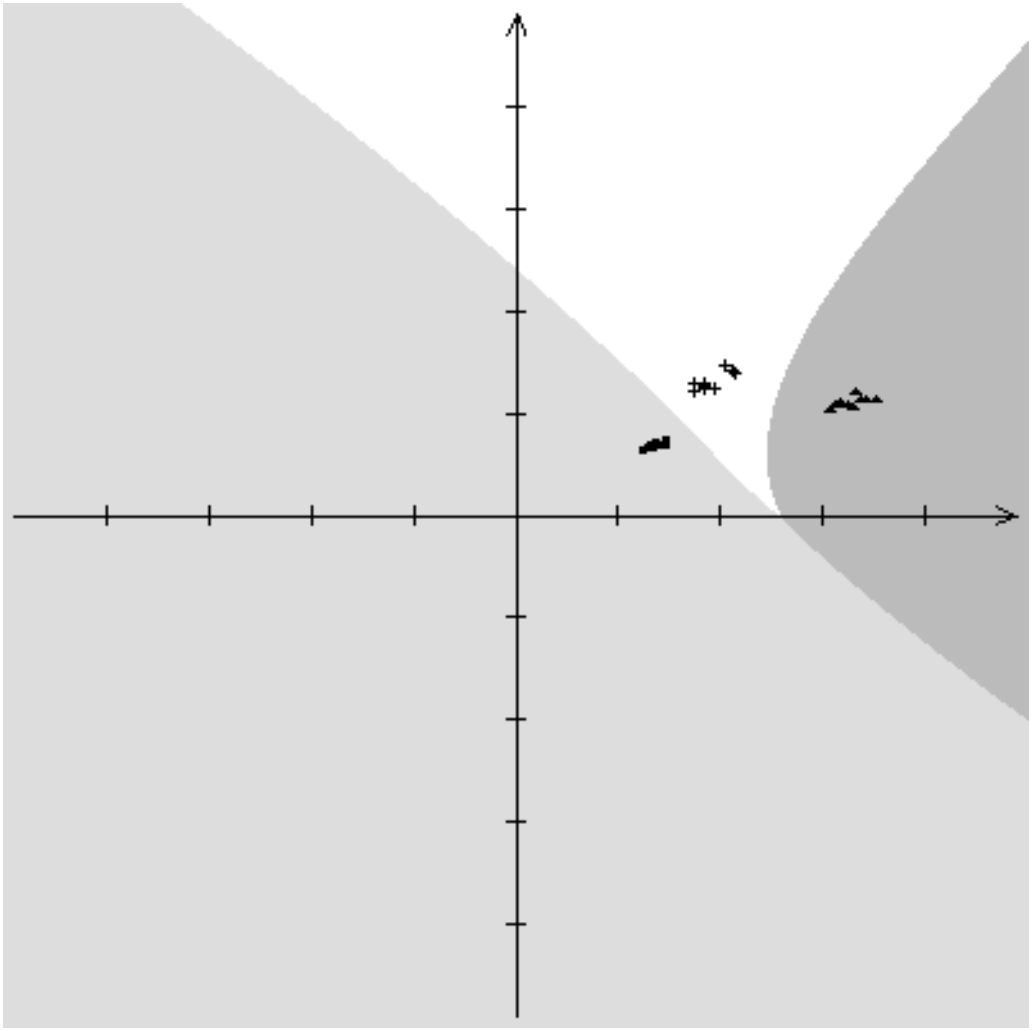


Figure 9: Classification results by the generating shrinking algorithm (scale-invariant) with decision regions superimposed on the training patterns.



After 2000 epochs

Figure 10: Classification results by the back-propagation method (three-layer perceptron, 10 neurons in the first layer, 10 in the second, and 3 in the third) with decision regions superimposed on the training patterns.

6 Conclusions

A novel training algorithm has been developed. It is fast, guaranteed to converge, and its generalisation behaviour is analytically known. With a small value for the reference number, which controls the generalisation behaviour, the algorithm achieves scale-invariant generalisation; with a large value for the reference number, the algorithm functions like a nearest-neighbour classifier, which shows that a three-layer feed-forward network is able to implement the nearest-neighbour rule.

The two spiral problem and a texture classification problem using the generating-shrinking algorithm and the back-propagation algorithm have been presented for comparison. Comparison results show (1) the generating-shrinking algorithm is substantially faster (several hundred times faster) than back-propagation. (2) generalisation of this algorithm is better than that of back-propagation. Experiments also show that the shrinking stage can greatly reduce the number of neurons. Although detrimental effects on generalisation by neuron elimination are observed in some cases, the shrunk nets still generalises better than the back-propagation approach.

References

[1] S. C. Ahalt, T. Jung, and A. K. Krishnamurthy. A comparison of radar signal classifiers. In *IEEE International Conference on Systems Engineering*, pages 609–612, 1990.

[2] Z. P. Lo and B. Bavarian. Comparison of a neural network and a piecewise linear classifier. *Pattern Recognition Letters*, 12:649–655, 1991.

[3] L. P. Cordella, C. De Stefano, F. Tortorella, and M. Vento. Improving character recognition rate by a multi-net neural classifier. In *Proc. 11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, pages 615–618, 1992.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representation by error propagation. In *Parallel Distributed Processing: Exploration of the Microstructure of Cognition*, volume 1, pages 318–362. The MIT Press, Cambridge, MA, 1986.

[5] J. F. Kolen and J. B. Pollack. Back-propagation is sensitive to initial conditions. *Complex Systems*, 4:269–280, 1990.

[6] C. Hall and R. Smith. Pitfalls in the application of neural networks for process control. In *IEE Control Engineering Series 46: Neural Networks for Control and Systems*, pages 243–256. Peter Peregrinus Ltd., London, 1992.

[7] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, 1987.

[8] X. Yang and F. T. S. Yu. Optical implementation of the hamming net. *Applied Optics*, 31:3999–4003, 1992.

[9] M. Mezard and J. P. Nadal. Learning in feed-forward layered networks: the tiling algorithm. *Journal of Physics. A: Mathematical and General*, 22:2191–2203, 1989.

[10] R. Zollner, H. J. Schmitz, F. Wunsch, and U. Krey. Fast generating algorithm for a general three-layer perceptron. *Neural Networks*, 5:771–777, 1992.

[11] P. A. Devijver and J. Kittler. *Pattern Recognition: a Statistical Approach*. Prentice Hall International, Engle Cliffs, New Jersey, 1982.

[12] T. Denoeux and R. Lengelle. Initializing back-propagation networks with prototypes. *Neural Networks*, 6:351–363, 1993.

[13] S. Sin and R. J. P. Defigueiredo. Efficient learning procedures for optimal interpolative nets. *Neural Networks*, 6:99–113, 1993.

[14] P. Brodatz. *Textures: a Photographic Album for Artists and Designers.* Reinhold, New York, 1968.

[15] W. K. Pratt. *Digital Image Processing*, chapter 17, pages 557–596. John Wiley & Sons, Inc, New York, 2nd edition, 1991.

Appendices

A. Proof of theorem 1

Lemma 1 *Two extended patterns* $p'_1 = (p_1, r)$ *and* $p'_2 = (p_2, r)$, $r \neq 0$ *are linearly dependent if and only if* $p_1 = p_2$.

Proof.

1. If $p_1 = p_2$ then $p'_1 = p'_2$, therefore p'_1 and p'_2 are linearly dependent.

2. If p'_1 and p'_2 are linearly dependent then we can find $a, b \in \mathcal{R}$, $|a| + |b| \neq 0$, such that $ap'_1 + bp'_2 = 0$ which implies

$$\begin{aligned} & (ap_1 + bp_2, ar + br) = 0 \qquad (9) \\ \Rightarrow \quad & \begin{cases} ap_1 + bp_2 = 0 \\ ar + br = 0 \end{cases} \\ \Rightarrow \quad & p_1 = p_2 \ . \quad \square \end{aligned}$$

The output of the k-th neuron in the first layer o_k^{L1} is said to be the maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$ if $o_k^{L1} \geq o_i^{L1}$ for $1 \leq i \leq n^{L1}$, $i \neq k$; o_k^{L1} is said to be the unique maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$ if $o_k^{L1} > o_i^{L1}$ for $1 \leq i \leq n^{L1}$, $i \neq k$.

Lemma 2 *If the training pattern* p_k *that is unique in the training set is inputed to the trained network, then* o_k^{L1} *is the unique maximum of* $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$.

Proof. By (2), we obtain

$$\begin{aligned} o_i^{L1} &= \sum_{j=1}^{n+1} w_{ij}^{L1} p'_{k,j} \text{ for } 1 \le i \le n^{L1} \\ &= w_i^{L1} \cdot p'_k \\ &\le \|w_i^{L1} \cdot p'_k\| \\ &\le \|w_i^{L1}\| \|p'_k\| \text{ (Schwarz's inequality)} , \\ \text{where} \quad & w_i^{L1} = (w_{i1}^{L1}, w_{i2}^{L1}, ..., w_{i(n+1)}^{L1}) . \end{aligned} \tag{10}$$

From (6), however

$$\|w_i^{L1}\| = \|\frac{p'_i}{\|p'_i\|}\| = 1 , \tag{11}$$

therefore

$$o_i^{L1} \le \|p'_k\| . \tag{12}$$

On the other hand

$$\begin{aligned} o_k^{L1} &= \sum_{j=1}^{n+1} w_{kj}^{L1} p'_{k,j} \\ &= w_k^{L1} \cdot p'_k \\ &= \frac{p'_k}{\|p'_k\|} \cdot p'_k \end{aligned} \tag{13}$$

$$= \|p'_k\| \ .$$

It follows that o_k^{L1} is the maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$.

Since p_k is unique in the training set, i.e., $p_i \neq p_k \quad \forall i \neq k$, then, by lemma 1, p'_i and $p'_k \quad \forall i \neq k$ are linearly independent. So that the equality of (12) is obtained if and only if $i = k$. Hence, o_k^{L1} is the unique maximum. □

Lemma 3 *If o_k^{L1} is the unique maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$, then the network output is $M(p_k)$.*

Proof. From (3) and (7), it is easily seen

$$o_i^{L2} = \begin{cases} 1 & \text{if } \ o_i^{L1} \ \text{ is the maximum of } \ \{o_i^{L1} : 1 \leq i \leq n^{L1}\} \\ 0 & \text{otherwise} \end{cases} \ . \tag{14}$$

By the antecedent, it follows

$$o_i^{L2} = \begin{cases} 1 & \text{if } \ i = k \\ 0 & \text{otherwise} \end{cases} \ , \tag{15}$$

by (4) and (8), therefore

$$o_i^{L3} = \begin{cases} 1 & \text{if } \ i = M(p_k) \\ 0 & \text{otherwise} \end{cases} \ , \tag{16}$$

which means the network output is $M(p_k)$. $\square$

Theorem 1 *The network constructed in the first phase is guaranteed to classify each training pattern correctly.*

Proof. By lemma 2, if the training pattern p_k is inputed to the trained network, then o_k^{L1} is the unique maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$. This unique maximum, by lemma 3, leads to network output of $M(p_k)$, thus proving the theorem. $\square$

B. The decision regions

Let $G_{ki} = \{p \in R^n : p \Rightarrow o_k^{L1} > o_i^{L1}\}$, $E_{ki} = \{p \in R^n : p \Rightarrow o_k^{L1} = o_i^{L1}\}$, $S_{ki} = \{p \in R^n : p \Rightarrow o_k^{L1} < o_i^{L1}\}$, $k \neq i$, where $p \Rightarrow o_k^{L1} > o_i^{L1}$ ($p \Rightarrow o_k^{L1} = o_i^{L1}$, $p \Rightarrow o_k^{L1} < o_i^{L1}$ respectively) means that the pattern p leads to $o_k^{L1} > o_i^{L1}$ ($o_k^{L1} = o_i^{L1}$, $o_k^{L1} < o_i^{L1}$ respectively) when it is inputed to the network.

From (2), one obtains

$$\begin{aligned} o_i^{L1} &= \sum_{j=1}^{n+1} w_{ij}^{L1} p'_j \\ &= w_i^{L1} \cdot p' \quad . \end{aligned} \tag{17}$$

It follows, therefore

$$
\begin{aligned}
& o_k^{L1} = o_i^{L1} \qquad (18)\\
\Rightarrow \quad & w_k^{L1} \cdot p' = w_i^{L1} \cdot p' \\
\Rightarrow \quad & (p'_k \|p'_i\| - p'_i \|p'_k\|) \cdot p' = 0 \quad .
\end{aligned}
$$

Substituting using $p'_i = (p_i, r)$, $p'_k = (p_k, r)$, $p' = (p, r)$, (18) gives

$$
(p_k \sqrt{p_i \cdot p_i + r^2} - p_i \sqrt{p_k \cdot p_k + r^2}) \cdot p + r^2 (\sqrt{p_i \cdot p_i + r^2} - \sqrt{p_k \cdot p_k + r^2}) = 0 \quad . \qquad (19)
$$

One sees that, in the variable of p, (19) is a hyper-plane in $\mathcal{R}^n$. i.e., $E_{ki} = \{p \in \mathcal{R}^n : p \Rightarrow o_k^{L1} = o_i^{L1}\}$ is a hyper-plane in $\mathcal{R}^n$.

Similar procedures give

$$
\begin{aligned}
& o_k^{L1} > o_i^{L1} \qquad (20)\\
\Rightarrow \quad & (p_k \sqrt{p_i \cdot p_i + r^2} - p_i \sqrt{p_k \cdot p_k + r^2}) \cdot p + r^2 (\sqrt{p_i \cdot p_i + r^2} - \sqrt{p_k \cdot p_k + r^2}) > 0 \quad ,
\end{aligned}
$$

and

$$
\begin{aligned}
& o_k^{L1} < o_i^{L1} \qquad (21)\\
\Rightarrow \quad & (p_k \sqrt{p_i \cdot p_i + r^2} - p_i \sqrt{p_k \cdot p_k + r^2}) \cdot p + r^2 (\sqrt{p_i \cdot p_i + r^2} - \sqrt{p_k \cdot p_k + r^2}) < 0 \quad .
\end{aligned}
$$

Whence $G_{ki} = \{p \in \mathcal{R}^n : p \Rightarrow o_k^{L1} > o_i^{L1}\}$ and $S_{ki} = \{p \in \mathcal{R}^n : p \Rightarrow o_k^{L1} < o_i^{L1}\}$ are two subsets (regions) of $\mathcal{R}^n$ divided by the hyper-plane $E_{ki} = \{p \in \mathcal{R}^n : p \Rightarrow o_k^{L1} = o_i^{L1}\}$.

As stated in lemma 3, if o_k^{L1} is the unique maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$ then the network output is $M(p_k)$. An equivalence to this is $o_k^{L1} > o_i^{L1} \quad \forall i \neq k$. Hence the region in which a pattern leads to o_k^{L1} being the unique maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$ and the network output being $M(p_k)$ is $H_k = \cap_{\forall i \neq k} G_{ki}$. From lemma 2, if p_k is the network input, then o_k^{L1} is the unique maximum of $\{o_i^{L1} : 1 \leq i \leq n^{L1}\}$. Hence $p_k \in G_{ki} \quad \forall i \neq k$, which implies $p_k \in H_k$. We summarise that for each p_k in the training set there is a region H_k containing p_k such that a pattern in H_k is classified into the $M(p_k)$-th class.

Naturally, there can be more than one training pattern falling into the same class. It is clear from the above discussions that a pattern in $H_k \quad \forall k \in \{k \in \mathcal{N} : M(p_k) = m\}$ causes the network output to be m, therefore the decision region of the m-th class is

$$\begin{aligned} D_m &= \bigcup_{k \in \{k \in \mathcal{N} : M(p_k) = m\}} H_k \\ &= \bigcup_{k \in \{k \in \mathcal{N} : M(p_k) = m\}} \bigcap_{i \neq k} G_{ki} \quad . \end{aligned} \tag{22}$$

C. Limiting behaviour

1. As r approaches 0.

Taking limit, (19) gives

$$(p_k\|p_i\| - p_i\|p_k\|) \cdot p = 0 \tag{23}$$
$$\Rightarrow \quad \frac{p_k}{\|p_k\|} \cdot p = \frac{p_i}{\|p_i\|} \cdot p \quad .$$

This is a hyper-plane containing the origin and having the property that if p is any point on the hyper-plane then $\angle p0p_k = \angle p0p_i$. i.e., the hyper-plane separates p_k, p_i in an equal-angle sense.

2. As r approaches ∞.

Dividing by r, (19) gives

$$(p_k\sqrt{\frac{p_i \cdot p_i}{r^2} + 1} - p_i\sqrt{\frac{p_k \cdot p_k}{r^2} + 1}) \cdot p + r(\sqrt{p_i \cdot p_i + r^2} - \sqrt{p_k \cdot p_k + r^2}) = 0 \quad . \tag{24}$$

Multiplying the numerator and denominator of the second term by

$$\sqrt{p_i \cdot p_i + r^2} + \sqrt{p_k \cdot p_k + r^2} \tag{25}$$

gives

$$(p_k\sqrt{\frac{p_i \cdot p_i}{r^2} + 1} - p_i\sqrt{\frac{p_k \cdot p_k}{r^2} + 1}) \cdot p + r\frac{p_i \cdot p_i - p_k \cdot p_k}{\sqrt{p_i \cdot p_i + r^2} + \sqrt{p_k \cdot p_k + r^2}} = 0 \quad . \tag{26}$$

Taking limit, gives

$$\begin{aligned} & (p_k - p_i) \cdot p + \frac{p_i \cdot p_i - p_k \cdot p_k}{2} = 0 \\ \Rightarrow \quad & (p_k - p_i) \cdot (p - \frac{p_k + p_i}{2}) = 0 \quad . \end{aligned} \tag{27}$$

This is a hyper-plane that is perpendicular to the line segment joining p_k and p_i, and bisects it. As a result, if p is any point on the hyper-plane then $\|p - p_k\| = \|p - p_i\|$. i.e., the hyper-plane separates p_k and p_i in an equal-distance sense.