IN-38
145564
P.17

NASA Technical Memorandum 105677

Self-Growing Neural Network Architecture Using Crisp and Fuzzy Entropy

Krzysztof J. Cios
Lewis Research Center
Cleveland, Ohio

and

Ohio Aerospace Institute
Brook Park, Ohio

Invited paper prepared for the
International Conference on Aerospace Sensing
sponsored by the Society of Photo-Optical Instrumentation Engineers
Orlando, Florida, April 20–24, 1992

SELF-GROWING NEURAL NETWORK ARCHITECTURE USING CRISP AND FUZZY ENTROPY

Krzysztof J. Cios*
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, OH 44135
and
Ohio Aerospace Institute
Brook Park, OH 44142

SUMMARY

The paper briefly describes the self-growing neural network algorithm, CID3, which makes decision trees equivalent to hidden layers of a neural network. The algorithm generates a feedforward architecture using crisp and fuzzy entropy measures. The results of a real-life recognition problem of distinguishing defects in a glass ribbon and of a benchmark problem of differentiating two spirals are shown and discussed.

INTRODUCTION

Supervised neural network algorithms usually require implementation of a trial-and-error method to find proper architectures. To help in determining a feedforward neural network architecture, it was shown [1] that a four-layered network with two hidden layers can solve arbitrary classification problems. Irie and Miyake [2] proved that a three-layered backpropagation network with an infinite number of nodes in the hidden layer can also solve arbitrary mapping problems. The "tiling" algorithm of Nadal [3] generates a feedforward network in a sequential manner by adding nodes and layers without the need for guessing the network's architecture, but it does not specify the sequence in which nodes should be added to maximize classification of training examples. The Algorithm described in [4] uses information entropy to determine generation of nodes and hidden layers.

Information entropy, however, has been used for a long time in machine learning research, where numerous learning algorithms have been developed to solve pattern recognition problems. A machine learning algorithm of particular interest to us is the ID3 algorithm of Quinlan [5], which dynamically generates a decision tree while minimizing information entropy. Recent studies of the ID3 algorithm and backpropagation neural networks [6, 7] prompted that the ideas similar to the ID3 algorithm may be used to answer two fundamental questions concerning a neural network architecture, namely, how to decide the number of layers and the number of nodes per layer. One goal of this paper is to show the close relationship between inductive machine learning and feedforward neural networks. This will be done by introducing the main ideas of a Continuous ID3 (CID3) algorithm [8].

Machine learning and neural networks are two very closely related fields of artificial intelligence, sharing many common ideas and problems. Effective methods of one can be used to overcome the difficulties of the other. It is interesting to note that the starting point in the development of the CID3

*On sabbatical leave from The University of Toledo.

algorithm was machine learning [9]. The advantage of using a machine learning approach to generate a feedforward neural network is that the knowledge embedded in the connections and weights can be translated into decision rules. To achieve fast convergence, a learning rule using information entropy was combined with Cauchy training [8, 10, 11] in the CID3 algorithm.

In order to make the main ideas of the CID3 algorithm clear, it is necessary to briefly introduce Quinlan's ID3 algorithm first [5]. ID3 generates decision rules from a set of training examples. Each example is represented by a list of features. In the training process, class memberships of the input data must be known. The idea is to find the minimum number of original features that suffice in determining class memberships. ID3 uses information theory to select features which give the greatest information gain or decrease of entropy. Entropy is defined as $-p\log_2 p$, where probability p is determined from the frequency of occurrence. Since the number of new nodes added to a decision tree depends on the number of values that a selected feature can take on, the ID3 algorithm requires features to have discrete values. The generated decision tree is then described in terms of hierarchical decision rules which must be used in the order specified by the tree structure. The condition part of a decision rule consists of a number of feature tests linked by and/or logical operators. The drawback of a feature test is that the correlations between features are ignored. ID3 considers only how significant an individual feature is for classifying training examples. The next section shows how an Adaline (adaptive linear neuron) [12] can be used for knowledge representation.

EQUIVALENCE OF A DECISION TREE AND A HIDDEN LAYER

As said before, the basic idea of the ID3 algorithm is to detect a feature yielding maximum information gain, so the training examples can be correctly classified. Let us consider a problem [8] of distinguishing nine positive examples from eight negative examples, as depicted in figure 1.

The difficulty of applying the ID3 algorithm, and calculating corresponding entropy, comes from the fact that the coordinates of x_1 and x_2 take on continuous values. In order to apply ID3 to this problem, one could use thresholds, so that the examples could be located within certain regions [13]. The thresholds may be represented as vertical and horizontal lines in a two-dimensional space. In real applications, however, decision regions are usually of higher order than a line, so the approximation may result in defining many high-dimensional decision regions. However, the decision region boundaries containing the same nine positive training examples can be formed by using hyperplanes defined by Adalines [12]. It is important to note that the feature test performed by ID3 can be treated as a special case of an Adaline with its hyperplane parallel to an axis. The decision region covering nine examples can be described by only three hyperplanes, as shown in figure 2.

To describe a decision region in terms of decision rules, the examples on the positive side of a hyperplane i (hyp_i) satisfy "$feature_i = 1$". Thus, the decision rules can be simply specified as follows:

IF $feature_1 = 0$, and $feature_2 = 0$, and $feature_3 = 1$, THEN class = positive.
IF $feature_2 = 1$, and $feature_3 = 0$, THEN class = positive.
IF $feature_1 = 1$, and $feature_2 = 1$, THEN class = positive.

Let us illustrate the conversion of a decision tree into a hidden layer by using the above example. First, if an example is tested on the positive side of hyp_1 then that example will be classified along edge 1, as shown in figure 3; otherwise, it will be classified along edge 0. Starting at the root node a, the training examples are divided into two nodes, b and c. At the second level of the decision tree, the examples from nodes b and c are tested against hyp_2. The examples on the positive side of the hyp_2 will be classified along edge 1 to a node descending from their parent node. Correspondingly, training examples on the negative side will be classified along edge 0 to the node descending from their parent node. The third hyperplane is needed to divide the examples at nodes f and g.

To convert the decision tree shown on the right-hand side of figure 3 into a hidden layer of a neural network, three Adalines are utilized. The directional vector of a hyperplane corresponds to the weight vector of an Adaline. For hyp_1, the weights w_1 and w_2 are the connection strengths of inputs x_1 and x_2 to Adaline #1 (neuron #1). Corresponding to the decision tree, a hidden layer with three nodes is generated, as shown on the left-hand side of figure 3.

CONTINUOUS ID3 ALGORITHM

In order to use ID3 for generating a neural network architecture, it was modified [8] to operate on continuous data and to search for the weight vectors. The following notation is used. There are N training examples, N^+ examples belonging to class "+" and N^- examples belonging to class "–". A hyperplane divides the examples as either lying on its positive (1) or negative (0) side. There are four possible outcomes:

N_1^+ number of examples from class "+" on side 1,
N_0^+ number of examples from class "+" on side 0,
N_1^- number of examples from class "–" on side 1, and
N_0^- number of examples from class "–" on side 0,

The following relations hold:

$$N = N^+ + N^- = N_1^+ + N_1^- + N_0^+ + N_0^- \tag{1a}$$

$$N_0^+ = N^+ - N_1^+ \tag{1b}$$

$$N_0^- = N^- - N_1^- \tag{1c}$$

At a certain level of a decision tree, it is assumed that N_r examples were divided by node r into: N_r^+ belonging to class "+" and N_r^- belonging to class "–". Relations analogous to those of equation (1) follow:

$$N_r = N_r^+ + N_r^- = N_{1r}^+ + N_{1r}^- + N_{0r}^+ + N_{0r}^- \tag{2a}$$

$$N_{0r}^+ = N_r^+ - N_{1r}^+ \tag{2b}$$

$$N_{0r}^- = N_r^- - N_{1r}^- \tag{2c}$$

The information entropy at level L of a decision tree is an average of entropies of all R nodes in this layer:

$$E = -\sum_{r=1}^{R} \frac{N_r}{N} \text{entropy}(L, r) \tag{3}$$

The change in information entropy is stated as [8]

$$\Delta E = -\sum_{r=1}^{R} \left[\frac{\partial E}{\partial N_{lr}^{+}} \Delta N_{lr}^{+} + \frac{\Delta E}{\Delta N_{lr}^{-}} \Delta N_{lr}^{-} \right] \tag{4}$$

The learning rule which minimizes the entropy function is

$$\Delta w_{ij} = -\rho \frac{\partial E}{\partial w_{ij}}$$

$$= -\rho \sum_{r=1}^{R} \sum_{i=1}^{N_r} \sum_{j=1}^{dim} \left\{ out_i (1 - out_i) x_j \left[D_i \left(\frac{\partial E}{\partial N_{lr}^{+}} - \frac{\partial E}{\partial N_{lr}^{-}} \right) + \frac{\partial E}{\partial N_{lr}^{-}} \right] \right\} \tag{5}$$

where D_i stands for the desired output of a training example, and out_i is a sigmoid function; r is a learning rate. The learning process for adjusting the weights can be stated in a vector form as follows:

$$W_{k+1} = W_k + \Delta W \tag{6}$$

Unfortunately, when the learning rule specified by equation (6) is used, the learning process might converge to a local minimum, since the gradient method does not guarantee constant information gain while generating a hidden layer. In order to increase the chance of finding the global minimum, the learning rule equation (6) was combined [8] with Cauchy training [10, 11].

$$\Delta w = T(t) \tan[\pi P(\Delta W \leq \Delta w) - \pi / 2] \tag{7}$$

To calculate the size of this weight change, a random number is selected from a uniform distribution over [0, 1], and substituted for $P(X \leq x)$. To determine whether to accept the weight change, Boltzmann distribution was used [8]. The probability of the error c is calculated by equation (8), where k is the Boltzmann constant.

$$P(c) = \exp\left(\frac{-c}{k_t} \right) \tag{8}$$

The final learning rule, incorporating the concept of a Cauchy training, is thus defined by equation (9), where random weight vector ΔW_{random} is calculated from equation (7), and η is a control parameter.

$$W_{k+1} = W_k + (1 - \eta) \Delta W + \eta \Delta W_{random} \tag{9}$$

So far, we have used only crisp entropy. As an alternative, the fuzzy entropy can be used also [14]. Comparison of the performance of the two measures is presented in the Results and Discussion section.

Let us now briefly introduce the notion of fuzzy entropy. A fuzzy entropy measure is a function f: P(X) -> R, where P(X) denotes the set of all fuzzy subsets of X. The function f assigns a value f(A) to each fuzzy subset A of X that characterizes the degree of fuzziness of A. It must satisfy the following three axioms:

f(A) = 0 if and only if A is a crisp set.
If A is less fuzzy than B, then f(A) ≤ f(B).
f(A) assumes the maximum if and only if A is maximally fuzzy.

DeLuca and Termini [15] were first to define the fuzzy entropy function:

$$f(A) = -\sum_{x \in X} \{\mu_A(x) \log_2 \mu_A(x) + [1 - \mu_A(x)] \log_2 [1 - \mu_A(x)]\}$$

where μ is a membership function.

Other measures of fuzziness were proposed by Kaufmann [16], Knopfmacher [17], and Loo [18]. The fuzzy entropy measure used here, which was also used in [14], was proposed by Kosko [19, 20]:

$$f(A) = \frac{\sum \text{count}(A \cap A^c)}{\sum \text{count}(A \cup A^c)} \tag{10}$$

where Σ count (sigma-count) is the fuzzy cardinality [21, 22] and A^c is a Zadeh's complement [23]. In the experiments reported in the Results and Discussion section, the fuzzy entropy equation (10) is used interchangeably with crisp entropy. The generalized fuzzy operations introduced by Dombi [24] are used to combine the fuzzy sets [14]. Generalized Dombi's operations form one of the several classes of functions which possess properties of fuzzy unions and intersections. The operations are detailed as follows.

$$\text{Fuzzy union} = \frac{1}{1 + \left[\left(\frac{1}{a} - 1\right)^{-\lambda} + \left(\frac{1}{b} - 1\right)^{-1}\right]^{-\frac{1}{\lambda}}} \tag{11}$$

where λ is a parameter by which different unions are distinguished, and $\lambda \in (0, \infty)$.

$$\text{Fuzzy intersection} = \frac{1}{1 + \left[\left(\frac{1}{a} - 1\right)^{\lambda} + \left(\frac{1}{b} - 1\right)^{\lambda}\right]^{\frac{1}{\lambda}}} \tag{12}$$

where λ is a parameter by which different intersections are distinguished, and $\lambda \varepsilon (0, \infty)$. Experience tells us that the parameter $\lambda = 4$ gives best results [25, 26].

Thus, in the learning rule equation (5), the fuzzy entropy f(A) can be used instead of crisp entropy E.

$$\Delta w_{ij} = -\rho \frac{\partial f(A)}{\partial w_{ij}} \tag{13}$$

where f(A) is a fuzzy entropy function. The grades of membership for fuzzy sets A and A^c were defined as follows [14]:

$$A = \left[\frac{N_{lr}^+}{N_{lr}}, \quad \frac{N_{lr}^-}{N_{lr}}, \quad \frac{N_{0r}^+}{N_{0r}}, \quad \frac{N_{0r}^-}{N_{0r}} \right] \tag{14a}$$

$$A^c = 1 - A \tag{14b}$$

Using mutual dependence of positive and negative examples on both sides of a hyperplane, the resulting fuzzy set A (with four grades of membership), and its fuzzy complement A^c, were expressed as

$$A = \left[\frac{N_r^- - N_{lr}^-}{N_r - N_{lr}^+ - N_{lr}^-}, \quad \frac{N_r^+ - N_{lr}^+}{N_r - N_{lr}^+ - N_{lr}^-}, \quad \frac{N_{lr}^+}{N_{lr}}, \quad \frac{N_{lr}^-}{N_{lr}} \right] \tag{15a}$$

$$A^c = \left[\frac{N_r^+ - N_{lr}^+}{N_r - N_{lr}^+ - N_{lr}^-}, \quad \frac{N_r - N_{lr}^- - N_r^+}{N_r - N_{lr}^+ - N_{lr}^-}, \quad \frac{N_{lr} - N_{lr}^+}{N_{lr}}, \quad \frac{N_{lr} - N_{lr}^-}{N_{lr}} \right] \tag{15b}$$

The four grades of membership will be used in equations (11) and (12) to calculate the fuzzy entropy equation (10). Obtained in this way, fuzzy entropy will be used to calculate the weights for the learning rule equation (9). The CID3 algorithm [8] follows:

Step 1. For a given problem with N training examples, follow the notations given in equations (1) and (2). Start with a random initial weight vector W_0.

Step 2. Utilize learning rule equation (9) and search for a hyperplane that minimizes the following entropy function (either crisp or fuzzy):

$$\min_{W_L^*} E = -\sum_{r=1}^{R} \frac{N_r}{N} \text{entropy}(L, r)$$

Step 3. If the minimized entropy is not zero, but is smaller than the previous value, add a node to the current layer and return to step 2. Otherwise, go to step 4.

Step 4. If the hidden layer consists of more than one node, generate a new layer that utilizes inputs from both the original training data and the outputs from all previously generalized layers, and go to step 2. If the hidden layer consists of only one node, then the problem is reduced to a linearly separable one; stop.

The CID3 algorithm was designed to generate a multiple layer network functioning like a single Adaline node and was defined as a super-Adaline [8]. To solve multiple-category classification problems, one can easily build a network [27] consisting of many such super-Adalines. After a hidden layer is generated by the CID3 algorithm, the outputs from all the generated hidden layers, together with the original inputs, are used to generate a new hidden layer. The use of the information from both the original training data and the outputs from the previously generated hidden layers allows a learning process to converge faster because of the increased dimensionality of training data [27]. The connection between non-adjacent layers are called shortcuts. Feedforward networks without shortcuts, like backpropagation, can be seen as a special case of such fully-connected networks with shortcuts.

From the machine learning point of view, a decision tree corresponds to a hidden layer of a neural network. If a correct classification of training examples is obtained, the corresponding entropy is reduced to zero. The learning which uses the knowledge from both original training examples and the outputs from hidden layers is actually a generalization process.

RESULTS AND DISCUSSION

In order to demonstrate the learning capability of the CID3 algorithm, it was applied to two problems. First, the CID3 algorithm was applied for recognition of defects found in manufactured glass [28] and compared with a standard backpropagation algorithm.

Several types of defects found in float glass ribbon can be grouped into two categories: actual defects, and surface anomalies. The latter are caused by water droplets or other airborne debris. The anomalies are detected as defects, and the section of glass containing them must be discarded, resulting in a loss of otherwise useable glass. Common defects found in flow glass ribbon [29] are described as follows:

True defects – permanent structures that degrade the homogeneity and optical quality of the glass.

Bubble – a round or elongated gaseous inclusion within the glass, which may be open at top or bottom surface.

Stone – a crystalline or amorphous inclusion within the glass, which may be opaque or slightly translucent.

Tin drop – a depression on the surface caused by a drop of molten tin adhering to the glass surface during forming; the solidified tin drop remains in the depression.

Surface anomalies – nonrejectable, temporary marks or spots on the glass surface.

Water droplet – a more or less hemispherical drop of liquid water, which may occur on either surface.

Water spot – mineral residue from a dried drop of water, which may occur on either surface.

A laser scanner system was used for the detection of defects in newly manufactured float glass ribbon. A rotating mirror scans a focused laser beam through the continually moving glass ribbon onto a sensor device. This device has two separate sensor arrays, one of which detects beam absorption, while the other detects beam refraction. The sensors create an electronic signal which varies voltage amplitude in proportion to the amount of beam intensity that is absorbed or refracted. The scanner traverses the moving glass ribbon with a focused laser beam 2400 times per second. This beam passes through the glass and is modified by any defects before striking the receiver. The receiver converts the incident light energy into a proportional electrical signal, which is the subject of our analysis.

A total of 293 images of defects were obtained [29]. Training examples consisted of 205 images selected randomly, while the remaining 88 were used as test examples. The sizes of the images obtained by the imaging system varied in proportion to the size of the actual defect. Typical images of a bubble and a water droplet (determined from several scans) are shown in figures 4(a) and 4(b). Images ranged in size from 30 by 20 pixels to 250 by 200 pixels. Because of this, preprocessing was done to normalize the images before they could be used to train a neural network. The preprocessing method [28] scaled the images to a 10 by 10 pixel frame without changing the aspect ratios or the image intensities. The scaled images were placed in the center of the 10 by 10 pixel frame as shown in figures 5(a) and 5(b). The 10 rows of an image, each 10 pixels in size, were then arranged to form a single vector.

For the purpose of distinguishing true defects (stone, bubble, or tin drop) in the glass from surface anomalies (water, water spot), the 205 training examples were divided into two groups representing defects and surface anomalies. The neural network was then trained with this data. The 88 test examples were then applied to the trained network to let it classify them into two categories. The correct recognition rates for the CID3 and backpropagation [29] networks are listed in table I. The results indicate that for two category classification, CID3 and backpropagation gave very high correct recognition rates for the test examples. The normalized CPU times required to train the networks are shown in table II. Training time required for backpropagation was much longer than that for the CID3 algorithm.

With backpropagation, the number of hidden layers and the number of nodes in each layer have to be determined. An inadequate number of layers or nodes might prevent convergence during training. An excessive number of nodes would result in a longer training time. The CID3 algorithm does not require the network architecture to be a priori specified. Based on the information entropy function, the algorithm adds the necessary number of layers and nodes to correctly recognize all the input-output pairs in the training data. The CID3 algorithm may be useful in situations where the networks are to be generated automatically and in real time. There may also be situations where there is a time constraint on the training time. Under these circumstances, the choice of the CID3 network would be appropriate.

Next, the CID3 algorithm was tested on difficult, non-linearly separable data [8]. The problem was to distinguish two spirals [8, 30]. The two sets of spiral data consisted of 192 points, with 96 points for each spiral. One spiral was generated as a reflection of another, namely $<x_1,y_1> = <-x_2,-y_2>$, which made the problem not linearly separable. The formulas used to generate the spirals are given below.

$$\rho = \alpha\theta, \quad \alpha = 10.0, \quad \theta \leq 6\pi$$

$$\text{spiral 1: } \begin{cases} x_1 = \rho\cos(\theta) \\ y_1 = \rho\sin(\theta) \end{cases} \qquad \text{spiral 2: } \begin{cases} x_2 = -\rho\cos(\theta) \\ y_2 = -\rho\sin(\theta) \end{cases}$$

The generated neural network architecture is shown in figure 6. Connections to the node in the second hidden layer are shown in detail, with connections to other layers shown by thick arrows. While generating a hidden layer, the corresponding decision tree is also recorded in order to specify a set of decision rules.

Comparison with other machine learning algorithms [5, 31, 32] that describe a concept by generating rectangular decision regions reveals the advantage of the CID3 algorithm — that it generates very concise descriptions. This contrasts with other machine learning algorithms which would generate many decision rules specifying numerous small rectangular regions for the two-spiral problem.

The obtained neural network architecture with the learned weights was applied to the spiral test data consisting of 150 by 150 pixels, specified in terms of x_1 and x_2 coordinates, that cover a square area of $[-15 \le x_1 \le 15, -15 \le x_2 \le 15]$. The result is shown in Figure 7(d). The white region represents spiral #1, and the black region represents spiral #2. Since at a hidden layer training examples are mapped into an image space by CID3, one may apply a machine learning algorithm to the output of a hidden layer and generate decision rules. The study of combining the CID3 algorithm and a machine learning algorithm called CLILP2 [9] was reported in [33]. Here we repeat the results in table III, and show the discriminating power of each network in figure 7.

The CLILP2 algorithm generates decision rules from the already extracted features much faster than CID3. This results in fast generation of a simple neural network architecture. No significant difference in discrimination ability was observed by analyzing the output images. This means that in the search for the optimal architecture, one may concentrate on the training time and the complexity of the network alone. It is easy to notice that Net_4 corresponds to the architecture shown in figure 5.

In order to demonstrate the performance of the fuzzy entropy measure, the CID3 was again applied to the spiral data using fuzzy entropy. Let us note here that learning to distinguish the two spirals is a very difficult task for backpropagation networks. This failure in training backpropagation neural networks was reported in [30] and was also confirmed by [33]. Actually, the CID3 algorithm can be seen as superior to work reported in [30], since the latter's architecture was obtained by using a trial-and-error method.

The fuzzy version of the CID3 algorithm generates the same architecture as the one generated by the crisp CID3. The nodes within the hidden layer are generated until the fuzzy entropy is reduced to zero. The crisp pseudoentropy measure accomplishes the same task quite well. However, a remarkable progress in terms of convergence time is achieved by using a fuzzy entropy measure with generalized Dombi operations.

CONCLUSIONS

CID3 self-generates a neural network architecture without the need to use a trial-and-error method to find an "optimal" architecture required by backpropagation-type networks. As a trade-off between the effort used for training and the quality of results, the CID3 algorithm seems to be competitive.

Unlike backpropagation, where correct classification of training examples is achieved only at the output layer, training examples are correctly recognized by CID3 at a hidden layer for which the information entropy is for the first time reduced to zero. In the process of generating a hidden layer by CID3, it is easy to specify the corresponding decision rules which describe the class memberships of the training examples [33].

The CID3 lends to machine learning algorithms its capability of working on continuous data and its immunity to noise. The CID3 algorithm helps in generalizing knowledge. The output of the last layer specifies the most general rule, and the outputs of a layer closer to the input layer specify more specific rules.

In conclusion, we have shown the advantages of the CID3 algorithm by illustrating the impact of a machine learning algorithm on the neural network algorithm in terms of what one can contribute to the other. Two alternative ways of calculating the entropy, crisp and fuzzy, were used. The fuzzy entropy method showed better performance in terms of convergent time.

ACKNOWLEDGEMENT

This work was partially supported by the National Science Foundation, Grant No. DDM-9015333.

REFERENCES

1. Lippmann, P.: An Introduction to Computing with Neural Nets. IEEE Mag. Acoust. Speech Signal Process, Apr. 1987, pp. 4–22.
2. Irie, B.; and Miyake, S.: Capabilities of Three-Layered Perceptions. IEEE International Conference on Neural Networks, IEEE, Piscataway, NJ, 1988, pp. I-641 – I-648.
3. Nadal, J.P.: New Algorithms for Feedforward Networks. Neural Networks and SPIN Glasses, World Scientific, 1989, pp. 80–88.
4. Bischel, M.; and Seitz, P.: Minimum Class Entropy: A Maximum Information Approach to Layered Networks. Neural Networks, vol. 2, 1989, pp. 133–141.
5. Quinlan, J.R.: Learning Efficient Classification Procedures and Their Application to Chess End-Games. Machine Learning: An Artificial Intelligence Approach, Vol. 1, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, eds., Tioga Publ., Palo Alto, CA, 1983, pp. 463–482.
6. Dietterich, T.G.; Hild, H.; and Bakiri, G.: A Comparative Study of ID3 and Back-propagation for English Text-to-Speech Mapping. Proceedings of the Seventh International Conference on Machine Learning, Morgan Kaufmann, Publ. San Mateo, CA, 1990, pp. 321–338.
7. Fisher, D.H.; and McKusick, K.B.: An Empirical Comparison of ID3 and Backpropagation. IJCAI–89: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publ., San Mateo, CA, 1989, pp. 788–793.
8. Cios, K.J.; and Liu, N.: A Machine Learning Method for Generation of a Neural Network Architecture: A Continuous ID3 Algorithm. IEEE Trans. Neural Networks, vol. 3, no. 2, 1992, pp. 280–291.
9. Cios, K.J.; and Liu, N.: CLILP2: A Machine Learning Algorithm Using Integer Linear Programming. In process, 1992.
10. Wasserman, P.D.: Combined Backpropagation/Cauchy Machine. Proceedings of the International Neural Network Society, Pergamon Press, 1988, pp. 125–138.
11. Szu, H.; and Hartley, R.: Fast Simulated Annealing. Phys. Lett. A, vol. 122, no. 8, 1987, pp. 157–162.
12. Widrow, B.; Winter, R.G., and Baxter, R.A.: Layered Neural Nets for Pattern Recognition. IEEE Trans. Acoust. Speech Signal Process., vol. 36, no. 7, 1988, pp. 1109–1118.
13. Quinlan, J.R.: Probabilistic Decision Trees. Machine Learning: An Artificial Intelligence Approach, vol. III, Y.K. Kodratoff and R.S. Michalski, eds., Tioga Publ., Palo Alto, CA, 1983, pp. 140–152.
14. Cios, K.J.; and Sztandera, L.: Continuous ID3 with Fuzzy Entropy Measures. IEEE International Conference on Fuzzy Systems, IEEE, New York, 1992, pp. 469–476.

15. DeLuca, A.; and Termini, S.: A Definition of a Nonprobabilistic Entropy in the Setting of Fuzzy Sets Theory. Inf. Control, vol. 20, no. 4, 1972, pp. 301–312.
16. Kaufmann, A.: Introduction to the Theory of Fuzzy Subsets. Academic Press, New York, 1975.
17. Knopfmacher, J.: On Measures of Fuzziness. J. Math. Anal. Appl., vol. 49, no. 3, 1975, pp. 529–534.
18. Loo, S.G.: Measures of Fuzziness. Cybernetica, vol. 20, no. 3, 1977, pp. 201–210.
19. Kosko, B.: Fuzzy Entropy and Conditioning. Inf. Sci., vol. 40, no. 2, 1986, pp. 166–174.
20. Kosko, B.: Neural Networks and Fuzzy Systems. Prentice Hall, 1992.
21. Zadeh, L.: A Computational Approach to Fuzzy Quantifiers in Natural Languages. Comput. Math. Appl., vol. 9, no. 12, 1983, pp. 149–184.
22. Zadeh, L.: The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems. Fuzzy Sets Syst., vol. 11, 1983, pp. 199–227.
23. Zadeh, L.: Fuzzy Sets. Inf. Control, vol. 8, no. 4, 1965, pp. 338–353.
24. Dombi, J.: A General Class of Fuzzy Operators, the DeMorgan Class of Fuzzy Operators and Fuzziness Measurements. Fuzzy Sets Syst., vol. 8, 1982, pp. 149–163.
25. Cios, K.J.; and Sztandera, L.M.: Generalized Fuzzy Operations to Diagnose Coronary Artery Stenosis. Submitted to Int. J. Approximate Reasoning, 1991.
26. Keller, J.M.; and Sztandera, L.M.: Special Relations Among Fuzzy Subsets of an Image. ASUNA '90: Uncertainty Modeling and Analysis, Proceedings of the Fifth International Symposium, IEEE, New York, 1990, pp. 207–211.
27. Nilsson, N.J.: The Mathematical Foundations of Learning Machines. Morgan Kaufmann Publ., San Mateo, CA, 1990.
28. Cios, K.J. et al.: Recognition of Defects in Glass Ribbons Using Neural Networks. Proc. of the 1991 National Science Foundation Design and Manufacturing Systems Conference, SME Publ., Dearborn, MI, 1991, pp. 203–206.
29. Cios, K.J. et al.: Study of Continuous ID3 and Radial Basis Function Algorithms for the Recognition of Glass Defects. International Joint Conference on Neural Networks, vol. 1, IEEE, New York, 1991, pp. 140–154.
30. Lang, K.; and Witbrock, M.J.: Learning to Tell Two Spirals Apart: Proceedings of the 1988 Connectionist Models Summer School, D. Touretzky, G. Hinton, and T. Sejnowski eds., Morgan Kaufmann, San Mateo, CA, 1988, pp. 52–59.
31. Pagallo, G.: Learning DNF by Decision Trees. IJCAI–89: Proceedings of Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publ., San Mateo, CA, 1989, pp. 630–644.
32. Michalski, R.S., et al.: The Multipurpose Incremental Learning System AQ15 and Its Testing Application to Medical Domains. AAAI–86: Proceedings Fifth National Conference on Artificial Intelligence, Morgan Kaufmann Publ., San Mateo, CA, 1986, pp.1041–1045.
33. Cios, K.J.; and Liu, N.: A Comparative Study of Machine Learning Algorithms for Generation of a Neural Networks. Artificial Neural Networks, T. Kohonen, et al., eds., vol. 1, Elsevier, New York, 1991, pp. 189–194.

TABLE I. – RECOGNITION RATES FOR CID3 AND BACKPROPAGATION NETWORKS

Method	True defect recognitions	Anomaly recognitions	Total
CID3 [a](100:7:6:6:1)	50/52 (96.15%)	35/36 (97.22%)	85/88 (96.59%)
Backpropagation [a](100:20:1)	51/52 (98.07%)	34/36 (94.44%)	85/88 (96.59%)

[a]Network architecture – the number of nodes in each layer.

TABLE II. – CPU TIMES TO TRAIN NETWORKS FOR TWO-CATEGORY CLASSIFICATION

Method	Normalized CPU time, minutes
CID3	161
Backpropagation	615

TABLE III. – TRAINING TIMES AND ARCHITECTURE PARAMETERS OF FOUR NETWORKS

Neural network	CPU time, minutes	Number of hidden layers	Total number of nodes	Number of nodes at layer 1	Number of nodes at layer 2	Number of nodes at layer 3	Number of nodes at layer 4	Number of nodes at layer 5	Number of nodes at layer 6
Net_1	19.57	2	47	24_{CID3}	22_{CLILP2}	1_{CLILP2}	----	----	----
Net_2	25.20	3	43	3_{CID3}	15_{CID3}	24_{CLILP2}	1_{CLILP2}	----	----
Net_3	36.90	4	31	3_{CID3}	15_{CID3}	5_{CID3}	7_{CLILP2}	1_{CLILP2}	----
Net_4	58.07	5	31	3_{CID3}	15_{CID3}	5_{CID3}	4_{CID3}	3_{CID3}	1_{CID3}

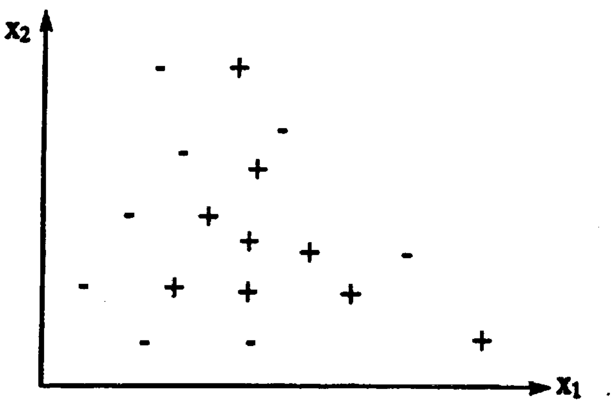


Figure 1. — Seventeen training examples belonging to class "+" and class "–".

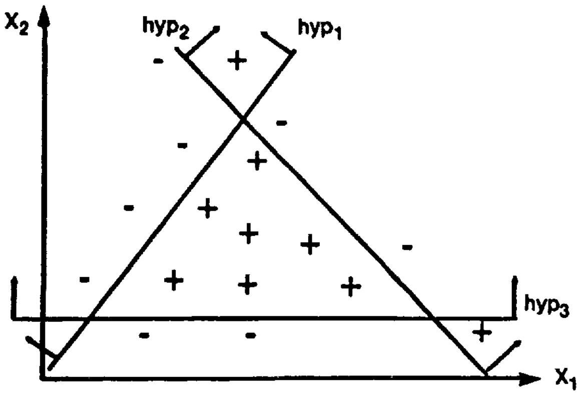
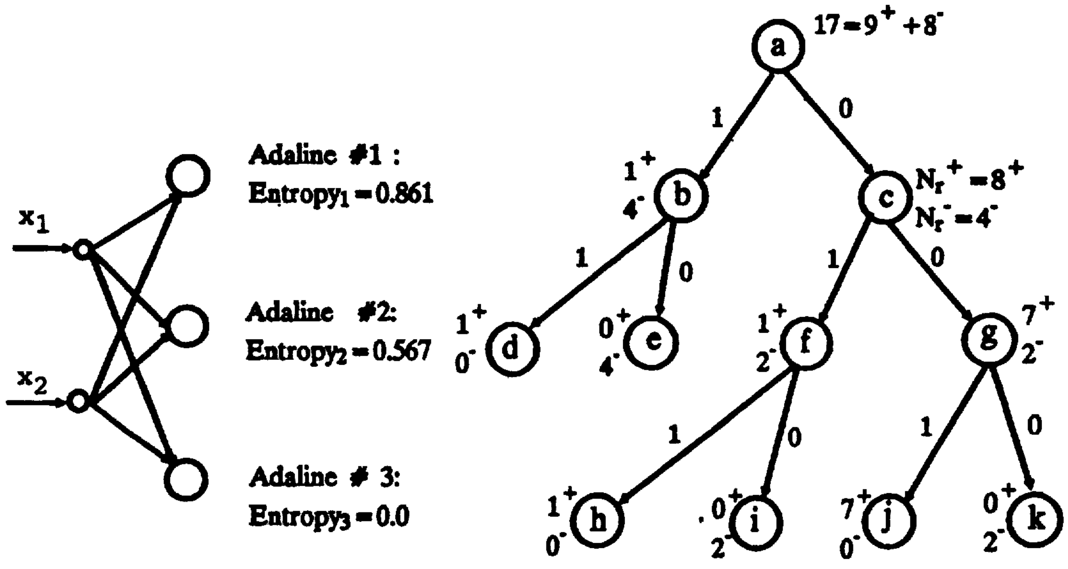


Figure 2. — Decision regions specified by hyperplanes.

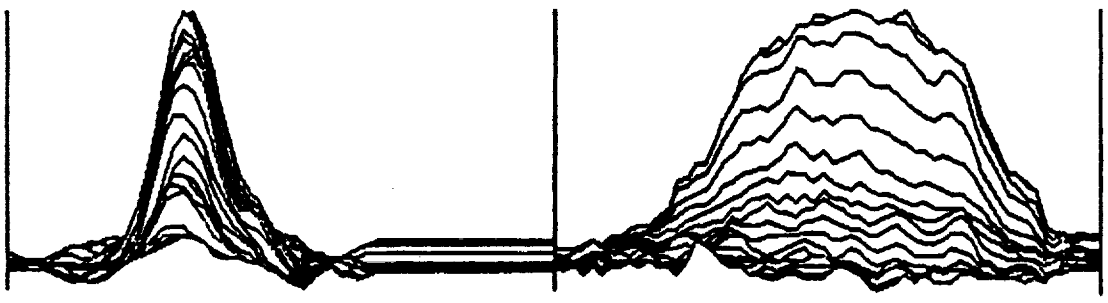


$$\text{Entropy}_1 = -\frac{1}{17}\left[\left(1\times\log_2\frac{1}{5} + 4\times\log_2\frac{4}{5}\right) + \left(4\times\log_2\frac{4}{12} + 8\times\log_2\frac{8}{12}\right)\right] = 0.861 \text{ bit}$$

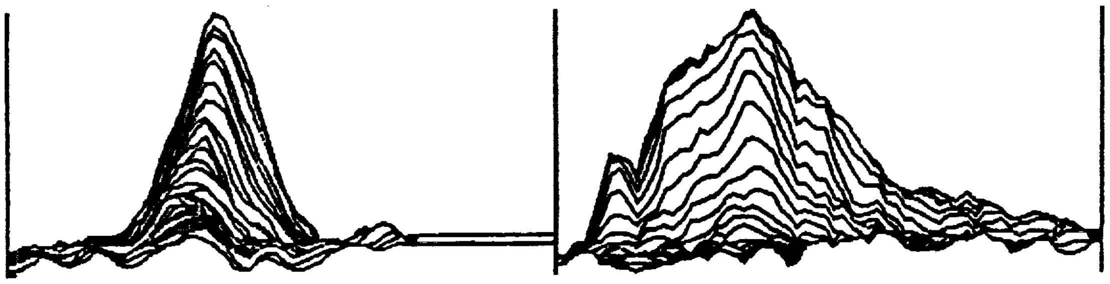
$$\text{Entropy}_2 = -\frac{1}{17}\left[(0+0) + (0+0) + \left(1\times\log_2\frac{1}{3} + 2\times\log_2\frac{2}{3}\right) + \left(7\times\log_2\frac{7}{9} + 2\times\log_2\frac{2}{9}\right)\right] = 0.567 \text{ bit}$$

$$\text{Entropy}_3 = -\frac{1}{17}\left[(0+0) + (0+0) + (0+0) + (0+0)\right] = 0.0 \text{ bit}$$

Figure 3. — Hidden layer corresponding to a decision tree and entropies calculated by using equation (3).

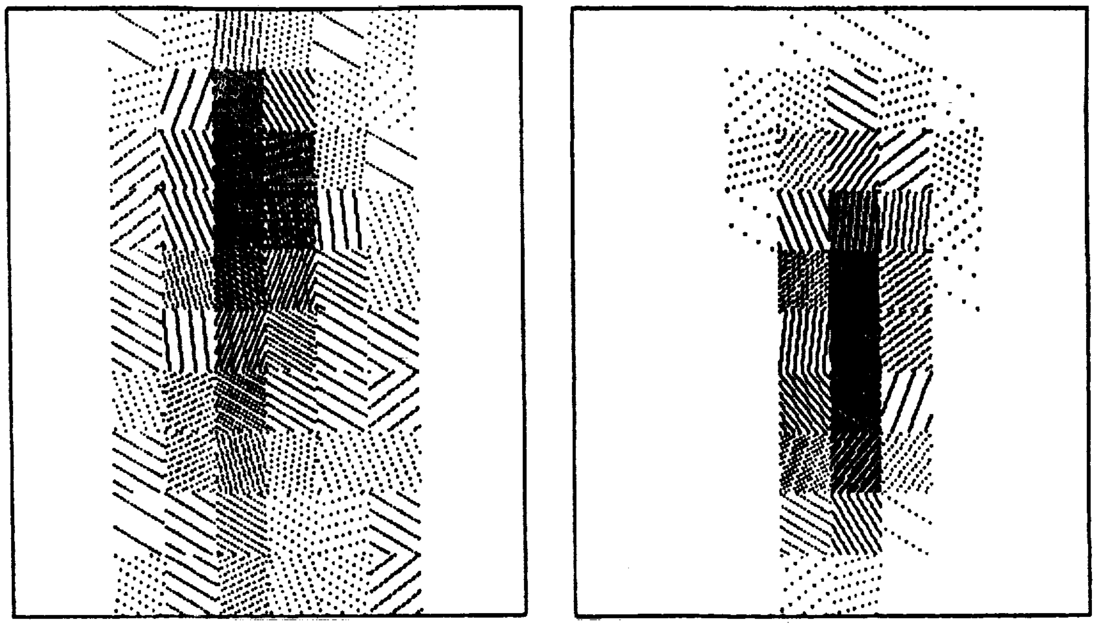


(a) Air bubble.



(b) Water droplet.

Figure 4. — Amplitude outlines of an air bubble and a water droplet.



(a) Air bubble. (b) Water droplet.

Figure 5. — Scaled images of an air bubble and a water droplet.

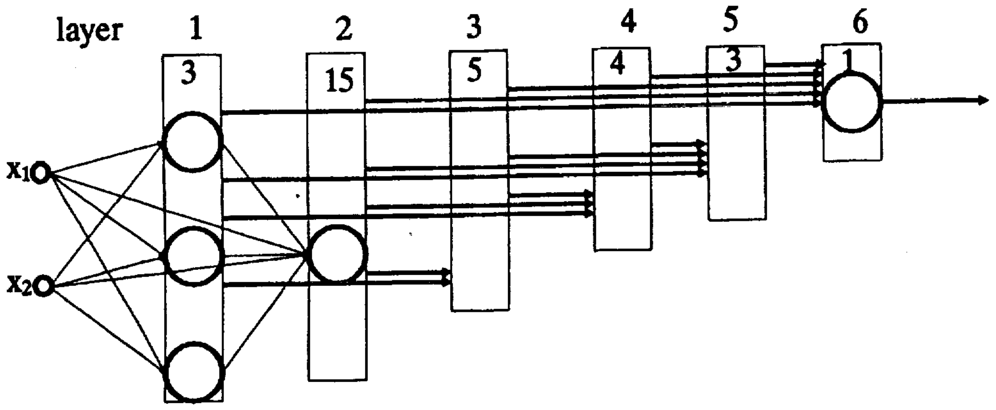


Figure 6. — Neural network architecture for distinguishing two spirals.

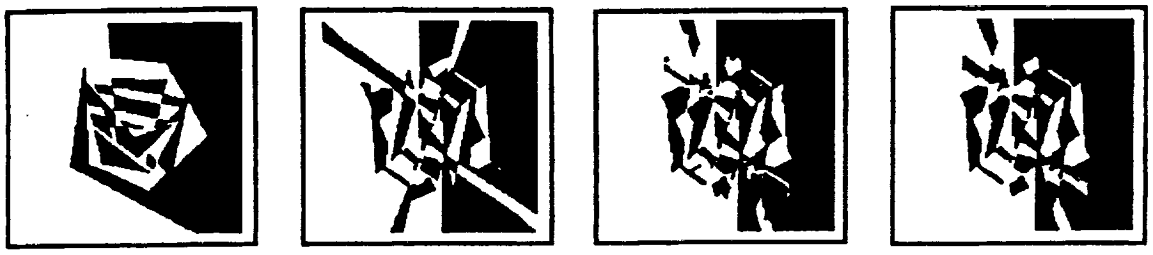


Figure 7.— Output images of four networks.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 1992	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE Self-Growing Neural Network Architecture Using Crisp and Fuzzy Entropy	5. FUNDING NUMBERS WU-510-01-50
6. AUTHOR(S) Krzysztof J. Cios	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	8. PERFORMING ORGANIZATION REPORT NUMBER E-7052
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-105677

11. SUPPLEMENTARY NOTES

Invited paper prepared for the International Conference on Aerospace Sensing sponsored by the Society of Photo-Optical Instrumentation Engineers, Orlando, Florida, April 20-24, 1992. Krzysztof J. Cios, NASA Lewis Research Center and Ohio Aerospace Institute, 2001 Aerospace Parkway, Brook Park, Ohio 44142. Mr. Cios is on sabbatical leave from the University of Toledo, Toledo, Ohio. Responsible person, Krzysztof J. Cios, (216) 433-3418.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 38	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The paper briefly describes the self-growing neural network algorithm, CID3, which makes decision trees equivalent to hidden layers of a neural network. The algorithm generates a feedforward architecture using crisp and fuzzy entropy measures. The results of a real-life recognition problem of distinguishing defects in a glass ribbon and of a benchmark problem of differentiating two spirals are shown and discussed.

14. SUBJECT TERMS Neural networks; Fuzzy logic; Nondestructive testing; Composites; Ultrasonics; Radiography; Material testing	15. NUMBER OF PAGES 18
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT