A Measure of Dependency in Feedforward Neural Networks

David K. Y. Chiu & Mike Y. W. Leung

Dept. Computing & Information Science
University of Guelph
Guelph, Ontario, Canada N1G 2W1

Email: dchiu@snowhite.cis.uoguelph.ca

Abstract

This paper presents an approach to measure relative dependency between input and output nodes (a feature selection problem) in different kinds of connected feedforward networks. The measure is based on the summation of the weighted paths between specified nodes. From the measure of dependency, we can identify the relevance or redundancy of the nodes involved. A statistical test is presented to provide an evaluation for comparison between the original network and the new network with node(s) deleted. The method can be extended to reduce the number of nodes in input and hidden layers. The method also provides useful information in the reconfiguration of the network architecture for better reliability and generalisability. The method is illustrated by applying to 2-dimensional shape classification problem using artificial and real images, and the parity-three function.

Keywords: feedforward networks, relative dependency, reconfigurable network, 2-D shape classification, parity-three function, node selection

1. Introduction

Measuring dependency relationships between variables is one of the basic tasks in data analysis. Numerous mathematical models have been successfully developed and applied. For example in statistics, regression analysis based on linear and non-linear relationships has been firmly established such that statistical relationship between dependent variables and independent variables can be modelled [4], [10]. In the linear case, the regression model provides reasonably accurate results reflecting their linear dependency. However in non-linear relationship, neural network often provides a practical framework for mathematical modelling [7]. A neural network can model a system on the basis of a set of training samples encoding the inputs and outputs [5]. With the development of the back-propagation algorithm in estimating the parametric weights, the feedforward networks become one of the most widely used network model to analyse given data. The network learns to map inputs into desired outputs by adaptive algorithms. If it is designed and trained properly, it can perform a very good generalisation in the context of a given architecture which models the behaviour of the domain. The multi-layer feedforward networks are also able to approximate formally all types of linear and non-linear functions [7], [5].

Consider the output nodes in a feedforward network, corresponding to the dependent (e.g., class) variables and the input nodes corresponding to a number of the independent (e.g., feature) variables of the data. The network then models the parallel distributed nature of the variables in a combined way under the constraints of the network architecture. When processing data, the value of each variable interacts with each other to produce the outputs. However, as in regression analysis, it is often useful to evaluate the dependency relationships between individual input and output variables, under the global context of the network.

The network is analogous to a "macro" relationship of the data, whereas the dependency relationship between individual input variable and the output variable is analogous to a "micro" relationship in the context of the global relationship. The significance of the relationships reinforces each other in a "closely coupled" manner to produce a better reliability of the model. This is the problem we are addressing here to designing ways to measure dependency between input and output and between hidden and output variables in feedforward neural networks.

2. Measure of dependency

A neural network can be defined as a 4-tuple <U, V, F, W> where U denotes the set of the nodes or processing units; V denotes the set of the connections; F denotes the output function for each node and W is the weight set such that there is a weight value corresponding to each connection. For a feedforward neural network, there is an input layer and an output layer, with a possibility of one or more hidden layers. The layers are labelled from the input layer to the output layer as (0), (1), ..., (n) such that the input layer has the label (0) and the output layer has the label (n). Each layer contains at least one node. The output of a node is calculated as the value of the function of the weighted sum of all its inputs. In feedforward networks, outputs of a node are allowed to connect (or feed) as inputs to nodes in a higher level layer. Normally, given a network, the set of U, V and the function F are fixed and the network can be denoted as $A(w)$, saying that the behaviour of the network in generating the outputs from the inputs depend only in the set of weights w.

Consider data $x = (x_1, x_2, ..., x_m)$ as inputs into a feedforward network with m nodes in the input layer. The node architecture and the connections between nodes from different layer are pre-defined as a model of the domain. Assuming a weight has been estimated through a training process for each connection. The outputs from the network can be calculated by "feeding" the values from the input layer towards the output layer. We are motivated by trying to answer the question that how much an output variable is dependent (linearly or non-linearly) on an input variable despite the distributed nature of the network in a given network given the set of estimated weights. One useful application of this is that we can use the measured quantity to evaluate the relevance of an input variable (or an intermediate node) on an output of a given network.

3. Two-layer networks with m-inputs and 1-output

Consider the simplest case where a two-layer feedforward network having m nodes in the input layer and one node in the output layer. The output value is denoted as $z = f(\sum_{j=1}^{m} w_j x_j)$ where f is defined as a non-linear differentiable monotonically non-decreasing function and x_j 's are the inputs. Since z is equally distributed among the inputs, a dependency relationship between x_j (> 0) and z can be calculated as follows:

$$I_{A(w)}(x_j, z) = \frac{|w_j x_j|}{\sum_{k=1}^{m} |w_k x_k|} z \qquad (1)$$

where $A(w)$ denotes the given feedforward network with the set of weights as $w = (w_1, w_2, ..., w_m)$, $\sum_{k=1}^{m} |w_k x_k| \neq 0$, and $|\ |$ denotes the absolute value. For convenience, we call this value $I_{A(w)}(x_j,z)$ the **I-value** between x_j and z. The higher the I-value is, the stronger the dependency of z on x_j. The absolute values indicate that the strength of the dependency does not depend on the sign of the weighted inputs, but the effect is the same whether they are positive or negative.

4. Multi-layer networks with m-inputs and p-outputs

Consider a weighted feedforward network of (n-1) hidden layers, without loss of generality, fully connected with inputs $x = (x_{1(0)}, x_{2(0)}, ..., x_{m(0)}) = (x_1, x_2, ..., x_m)$ and outputs $z = (z_{1(n)}, z_{2(n)}, ..., z_{p(n)}) = (z_1, z_2, ..., z_p)$ where the indices inside the brackets of the subscripts denote the layer number. Assuming that the outputs of z are normalised to be between 0 and 1. Let the number of nodes at layer l be $p(l)$, $l = 1, 2, ..., n$. Let the weights of the network between the kth node at layer (l-1) and the hth node at layer l be denoted by $w_{k(l-1)}^{h(l)}$, ($l = 1, 2, ..., n$). The weight set

$$w = \left\{ w_{k(l-1)}^{h(l)} \mid l = 1,2,...,n; 1 \le h(l) \le p(l); 1 \le k(l-1) \le p(l-1); w_{k(l-1)}^{h(l)} \neq 0 \right\}$$

includes all the weights such that there is a weight for each connection between layers. The output from the kth node in the layer (l-1) is $y_{k(l-1)}$. For a network that is not fully connected, it behaves as if the weight $w_{k(l-1)}^{h(l)} = 0$.

Consider a path between a given output node and an input node represented by their respective variables z_i and x_j. Denote the nodes traversed in the non-zero weighted path by their output values. The path from input node x_j can be represented as $\left(x_j, y_{h_1(1)}, ..., y_{h_{n-2}(n-2)}, y_{h_{n-1}(n-1)}, z_i\right)$ where the subscripts of h represent a particular node in the hidden layer chosen.

Consider a node in the path, the output value is equally dependent on all its inputs, after considering the corresponding weights, because of the additive operation. Therefore the relevance of each input can be calculated by the relative ratio of the absolute weighted values. A measure of dependence between the output z_i on x_j is then defined based on the summation of the absolute relative weights of all the paths connecting the two nodes. That is, the value z_i is distributed to all the non-zero weighted paths leading to it and the dependency corresponds to the calculated sum of all the paths between the two nodes. Mathematically, it is defined as:

$$I_{A(w)}(x_j, z_i) = \sum_{\forall h_{n-1}} \sum_{\forall h_{n-2}} \cdots \sum_{\forall h_1} \left\{ z_i \left(\frac{\left| w^{i(n)}_{h_{n-1}(n-1)} y_{h_{n-1}(n-1)} \right|}{\sum_k \left| w^{i(n)}_{k(n-1)} y_{k(n-1)} \right|} \right) \right.$$

$$\left(\frac{\left| w^{h_{n-1}(n-1)}_{h_{n-2}(n-2)} y_{h_{n-2}(n-2)} \right|}{\sum_k \left| w^{h_{n-1}(n-1)}_{k(n-2)} y_{k(n-2)} \right|} \right) \cdots \left(\frac{\left| w^{h_2(2)}_{h_1(1)} y_{h_1(1)} \right|}{\sum_k \left| w^{h_2(2)}_{k(1)} y_{k(1)} \right|} \right) \left(\frac{\left| w^{h_1(1)}_{j(0)} x_j \right|}{\sum_k \left| w^{h_1(1)}_{k(0)} x_k \right|} \right) \left. \right\} \quad (2)$$

for

$$\sum_k \left| w^{j(n)}_{k(n-1)} y_{k(n-1)} \right|, \sum_k \left| w^{h_{n-1}(n-1)}_{k(n-2)} y_{k(n-2)} \right|, \ldots, \sum_k \left| w^{h_2(2)}_{k(1)} y_{k(1)} \right|, \sum_k \left| w^{h_1(1)}_{k(0)} x_k \right| \neq 0.$$

Note that the numerators inside the curly bracket correspond to the weighted path between the input node x_j and the output node z_i. That is, a path connecting the nodes corresponding to the outputs of $(x_j, y_{h_1(1)}, \ldots, y_{h_{n-2}(n-2)}, y_{h_{n-1}(n-1)}, z_i)$. For a fully connected feedforward network, there are altogether $(p(1)\times p(2)\times \ldots \times p(n-1))$ unique paths between the two nodes, and hence there are altogether $(p(1)\times p(2)\times \ldots \times p(n-1))$ product terms in the summation. In addition, by definition, if the numerator or the denominator are zeros in a division, this particular path is ignored as it means the two nodes are not dependent in any way for this case. The rationale of the above formula is that, firstly an output value z_i is distributed back to an input x_j according to the number of paths between an input node and an output node. Secondly, it is proportional to the ratio of the absolute value of the weighted term to the weighted sum of the inputs in each node that makes up the steps of the path. This idea based on proportion of the weighted paths can be extended to other feedforward networks as long as the weighted inputs in each node are additive.

5. Estimating I-value from sampling data set

Given a set of sampling data $S = \{(x,\zeta)\}$ and a feedforward network $A(w)$, where $\zeta = (\zeta_1, \zeta_2, \ldots, \zeta_p)$, $0 \le \zeta_i \le 1$, are the expected outputs and $x = (x_1, x_2, \ldots, x_m)$ are the corresponding inputs using the network $A(w)$. We can evaluate the dependency between an input variable and an output variable by inputting each data x into the network. Values will be generated for the output nodes for each sampling data set. Since the zero vector sample has no effect on the training process, it is assumed that all zeros in such samples are defined to be small values or these samples are ignored. Notice that for a fixed value of x_j, a different I-value may be calculated for different n-vector x, because the outputs from the network is affected by other values in x globally as well. For training data set composes of data from p non-intersecting classes, say $S = S_1 \cup S_2 \cup \ldots \cup S_p$, $(S_i \cap S_j = \varnothing$ for $i \neq j)$, the expected outputs from the network can be designed such that $\zeta_i = 1$ for data from the ith class and $\zeta_i = 0$ otherwise. The sum of all ζ_i is equal to 1. To evaluate the dependency based on sampling data, we modify formula (2) considering the value of error as well. For convenience of notation, given a sample data $x \in S$, the estimate of correctness for z_i from the network with x as inputs can be calculated as $c_i = 1 - e_i = 1 - |\zeta_i - z_i|$. The value of error, denoted as e_i, where $0 \le e_i = |\zeta_i - z_i| \le 1$, is the absolute deviation of the calculated output from the expected output at the ith output node given data x. If $\zeta_i = 1$, then $c_i = z_i$ and if $\zeta_i = 0$, then $c_i = 1 - z_i$. We call this the **Sampling I-value** between x_j and z_i. When $\zeta_i = 1$, the I-value reflects the dependency to node i (or class i). When $\zeta_i = 0$, the I-value reflects the dependency to nodes other than i. It turns out that the formula in calculating the sampling I-value is the same for both cases and is defined as follows:

$$\hat{I}_{A(w)}(x_j, c_i) = \frac{1}{|S|} \sum_s I_{A(w)}(x_j, c_i), \quad \text{where } x_j \in x \in S,$$

and,

$$I_{A(w)}(x_j, c_i) = \sum_{\forall h_{n-1}} \sum_{\forall h_{n-2}} \cdots \sum_{\forall h_1} \left\{ c_i \left(\frac{\left| w^{i(n)}_{h_{n-1}(n-1)} y_{h_{n-1}(n-1)} \right|}{\sum_k \left| w^{i(n)}_{k(n-1)} y_{k(n-1)} \right|} \right) \right.$$

$$\left(\frac{\left| w^{h_{n-1}(n-1)}_{h_{n-2}(n-2)} y_{h_{n-2}(n-2)} \right|}{\sum_k \left| w^{h_{n-1}(n-1)}_{k(n-2)} y_{k(n-2)} \right|} \right) \cdots \left(\frac{\left| w^{h_2(2)}_{h_1(1)} y_{h_1(1)} \right|}{\sum_k \left| w^{h_2(2)}_{k(1)} y_{k(1)} \right|} \right) \left(\frac{\left| w^{h_1(1)}_{j(0)} x_j \right|}{\sum_k \left| w^{h_1(1)}_{k(0)} x_k \right|} \right) \left. \right\} \quad (3)$$

for

$$\sum_k \left| w^{i(n)}_{k(n-1)} y_{k(n-1)} \right|, \sum_k \left| w^{h_{n-1}(n-1)}_{k(n-2)} y_{k(n-2)} \right|, \ldots, \sum_k \left| w^{h_2(2)}_{k(1)} y_{k(1)} \right|, \sum_k \left| w^{h_1(1)}_{k(0)} x_k \right| \neq 0.$$

Equation (3) is similar to equation (2), except that c_i replaces z_i. The values of y's represent the corresponding outputs of the nodes in the network when the inputs are x's. The amount of calculated training error for z_i that can be contributed to x_j is estimated by the following measure as:

$$\hat{E}_{A(w)}(x_j, e_i) = \frac{1}{|S|} \sum_s I_{A(w)}(x_j, e_i),$$

where $x_j \in x \in S$, and,

$$E_{A(w)}(x_j, e_i) = \sum_{\forall h_{n-1}} \sum_{\forall h_{n-2}} \cdots \sum_{\forall h_1} \left\{ e_i \left(\frac{\left| w^{i(n)}_{h_{n-1}(n-1)} y_{h_{n-1}(n-1)} \right|}{\sum_k \left| w^{i(n)}_{k(n-1)} y_{k(n-1)} \right|} \right) \right.$$

$$\left. \left(\frac{\left| w^{h_{n-1}(n-1)}_{h_{n-2}(n-2)} y_{h_{n-2}(n-2)} \right|}{\sum_k \left| w^{h_{n-1}(n-1)}_{k(n-2)} y_{k(n-2)} \right|} \right) \cdots \left(\frac{\left| w^{h_2(2)}_{h_1(1)} y_{h_1(1)} \right|}{\sum_k \left| w^{h_2(2)}_{k(1)} y_{k(1)} \right|} \right) \left(\frac{\left| w^{h_1(1)}_{j(0)} x_j \right|}{\sum_k \left| w^{h_1(1)}_{k(0)} x_k \right|} \right) \right\} \quad (4)$$

for $\sum_k \left| w^{i(n)}_{k(n-1)} y_{k(n-1)} \right|, \sum_k \left| w^{h_{n-1}(n-1)}_{k(n-2)} y_{k(n-2)} \right|, \ldots, \sum_k \left| w^{h_2(2)}_{k(1)} y_{k(1)} \right|, \sum_k \left| w^{h_1(1)}_{k(0)} x_k \right| \neq 0.$

In feature selection such as forward or backward selection [8], a feature can be selected based on its I-value which represents the relative dependency given an output variable on an input variable. The magnitude of the I-value indicates how relevant a feature $x_j \in x$ is among all the inputs in determining the outputs in a network.

6. Using statistical test in network pruning

The choice of the number of nodes, for a given problem, often presents an interesting challenge. The calculation of the I-values can also be extended to the evaluation of the nodes. Here, we calculate the I-values of the paths from a node to be considered. We then construct a new network by deleting the node with the lowest I-values while keeping the rest of the parameters (or weights) unchanged. That is, without re-training the network, we use the same set of weights for comparison. After calculating the I-values, we proceed to compare the ones in the new network using the newly generated outputs in the context of the same parameters. A statistical t-test is used to evaluate whether the I-values of the original network is the same as the new one. The test hypothesis is given by:

H_0: $I_{A(w)} = I_{A(w)'}$ and

H_1: $I_{A(w)} \neq I_{A(w)'}$

where $A(w)$ is the original network and $A(w)'$ is the new network with the lowest I-value node deleted. For all i samples, we calculate $d_i = I_{A(w)} - I_{A(w)'}$. The test statistic t is defined as follows:

$$t = \frac{\sqrt{n_s}(\bar{d} - 0)}{S_d} \quad (5)$$

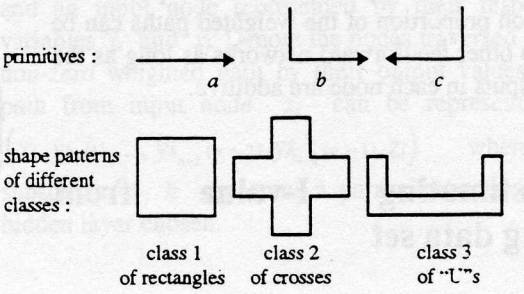
where $\bar{d}$ is the mean difference of d_i, S_d the standard deviation and n_s the total number of samples. It is a two-tailed test and the null hypothesis will be rejected if $t < -t_{\alpha/2, n_s - 1}$ or $t > t_{\alpha/2, n_s - 1}$ for certain pre-defined confidence level α with $(n_s - 1)$ degree of freedom. If the result is positive, there will be a strong evidence that the node is redundant and the deletion is correct while keeping the reliability of the network within the desired error level.

7. Experiments

A. Experiment 1 : Dependency between class and string primitives in 2-D shape data

In this experiment, we analysed the dependency in a network that classified two-dimensional patterns based on their shape represented as sequences of primitives as circular strings [6], [1], [2], (Figure 1).

Fig. 1 Shape patterns and the primitives



For example, a square could be represented by the circular string *abababab* and a rectangle could be represented by *aababaabab* (Figure 2a) and the data is shown in Figure 2b. The shape patterns could be evaluated by the weighted values from the primitives.

Fig. 2a Example of shapes represented by circular strings

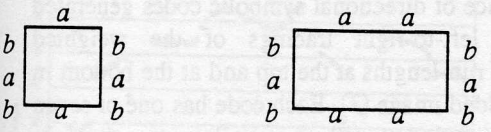


Fig. 2b Data used in experiment 1

Sample	Class	Circular string representation
1	1	*abaababaab*
2	1	*baaabaaabaaabaaa*
3	1	*aaaababaaaabab*
4	1	*abababab*
5	2	*baacabaabacababaaacababacaba*
6	2	*acababacababacababacabab*
7	2	*caabaaaabaacaababaacaabaaabaacaababa*
8	3	*babaaacaaaacaabaabaaaabaaaaaaaaabaaaaa*
9	3	*abcacbabaaabaaabaaaaabaaab*
10	3	*babacaaaaaaacababaaabaaaaaaaaaabaaa*

Assuming that each primitive type had the same set of corresponding weights, the inputs corresponding to the number of each primitive type observed in a string. The network was designed to have three input nodes corresponding to the number of each primitive observed in a string. It had two hidden layers of two nodes each, and three output nodes representing three classes of the shape pattern. The network represented a weighted relationship between the class and the primitive components of a string. There were nine possible pairs of relationships between the input and output variables. The nine relationships were the pairwise dependency between the three input primitive types variables and three output class variables. The outputs and the data from the nodes were generated through training based on the weights identified. The sampling I-values were then calculated when the network was stabilised and accepted. All the classes have the highest sampling I-values corresponding to primitive *b* (i.e., x_2). That means the classification process with the generated weights in the network has the highest dependency on primitive *b* comparing to other primitives (Table 1). Whereas primitive *a* (i.e., x_1) has the lowest sampling I-value, or the lowest dependency. From the data (Figure 2b), the number of *a* in a string varies which is consistent with the low I-values. The entry of the I-value in $\hat{I}_{A(w)}(x_3, z_i)$ indicates that there is no path from the specific input node to the output node. The result indicates the magnitude of dependency is consistent with the properties of the classes of rectangles, crosses, and "U"-shape objects.

Table 1. Sampling I-values of experiment 1

Class	$\hat{I}_{A(w)}(x_1, z_i)$ (primitive *a*)	$\hat{I}_{A(w)}(x_2, z_i)$ (primitive *b*)	$\hat{I}_{A(w)}(x_3, z_i)$ (primitive *c*)
Class 1 of rectangles (i=1)	0.1437	0.8504	-
Class 2 of crosses (i=2)	0.1271	0.3681	0.2875
Class 3 of "U"s (i=3)	0.0847	0.1645	0.1081

Note: '-' indicates no path from data generated

B. Experiment 2: Parity-three function

The parity-three function is a generalisation of the exclusive-OR function [9]. It includes three variables which map to values of 0 or 1.

(i) Experiment 2a: Parity-three function with 3 input nodes

Consider the relationship of the parity-three function, $z = f(x_1, x_2, x_3)$, where its values depend on all the independent variables of x_1, x_2, x_3 [9]. When $f(x_1, x_2, x_3) = 1$, the expected output value is $\zeta_i = 1$. Similarly, when $f(x_1, x_2, x_3) = 0$, the expected output value is $\zeta_i = 0$. We used a 3-layer network of three input nodes, three hidden nodes and one output node to train the data. The sampling I-values are calculated and described in Table 2. Since theoretically the parity-three function depends equally on all the independent variables, their I-values are expected to be the same for all x_1, x_2, and x_3, taken into the consideration the estimated nature of the weights in the network. This result is also observed in our experiment (Table 2, row of original network).

(ii) Experiment 2b: Parity-three function with an added variable

Next we introduced a dummy variable $x_r = \{0,1\}$ into the function denoted as $f(x_1, x_2, x_3, x_r)$, so that the function values did not depend on x_r. For each case of (x_1, x_2, x_3) values, two samples were created with x_r either 0 or 1. The sample size was increased to 16. We used a 3-layer network of four input nodes, three hidden nodes and one output node in this analysis. After training, the sampling I-values were calculated (Table 2, row of network with x_r). The experiment was repeated by adding one hidden node in a 3-layer network of four input nodes, one output nodes and four hidden nodes. The sampling I-values are described in Table 2 (row of network with x_r & added hidden node). In both experiments, the I-values for x_1, x_2, x_3 and the output do not change significantly, indicating that the function depends similarly on the independent variables x_1, x_2, x_3. However, the variable x_r generated very low I-values relatively (0.2505 and 0.2676) indicating the function or the output does not depend on x_r. Notice that the number of nodes in the hidden layer does not affect the calculation of the I-values (row 2 and 3 in Table 2). What is the behaviour of the I-value if a node such that the function completely depends is added? We demonstrated by adding a variable x_d which had the same values as the actual output value. The experiment was repeated with a 3-layer network with four input nodes, four hidden nodes and one output node. The experiment shows that variable x_d has very high I-values as compare to the other three variables (row 4 of Table 2). In another words, now, the output depends strongly on this newly added variable.

Table 2. Sampling I-values for parity-three network data

	$\hat{I}_{A(w)}(x_1, z)$	$\hat{I}_{A(w)}(x_2, z)$	$\hat{I}_{A(w)}(x_3, z)$	$\hat{I}_{A(w)}(x_r, z) / \hat{I}_{A(w)}(x_3, z)$
original network	0.4994	0.4995	0.4996	N/A
network with x_r	0.4994	0.4993	0.4993	0.2505
network with x_r & added hidden node	0.4993	0.4993	0.4925	0.2676
network with x_d	0.0824	0.0300	0.0665	0.9098

Note: N/A means not applicable

C. Experiment 3: Dependency in classification of biomolecular images

The data used in this experiment were taken from 103 biomolecular images of three classes [3]. The shape of a molecule in the image was represented by a sequence of directional symbolic codes generated from two left-to-right tracings of the weighted averages of run-lengths at the top and at the bottom in the thresholded image [3]. Each code has one of seven possible directional values similar to that in experiment 1 and each code at a position represented the rounded angle between the current and the next trace steps. The first seven code values represent the angle in the tracing in the upper portion of the image and the last seven code values represent that in the lower tracing. The 14 possible code values were arranged to correspond to the input nodes of the network similar to that in experiment 1 (Figure 3a and 3b). The analysis using our proposed method will indicate how a class depends on a directional code.

Fig. 3a An example of the thresholded image

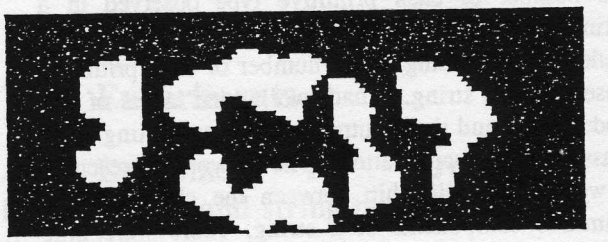
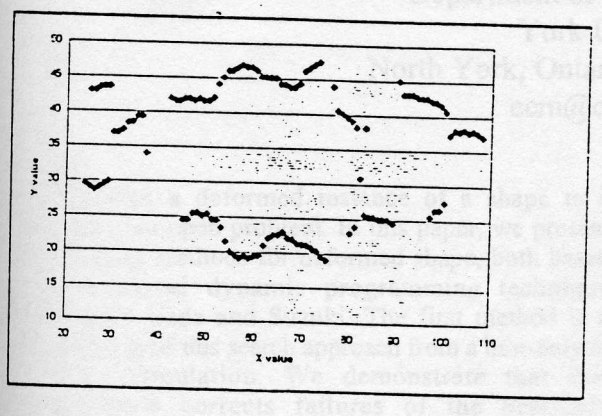


Fig. 3b The positions of weighted averages derived by run-length encoding of the image in 3a. A sequence of symbolic codes are then generated from tracing the points.



The network was then designed to have 14 input nodes, two hidden layers of three nodes each and three output nodes. The outputs represented three supervised classes of the objects. There were 42 (3×14) possible pairs of relationships between the input and output variables. The outputs from the node were generated after the network was trained. The sampling I-values were then calculated. The calculation indicates the upper downward tracing (directional code 6) and lower part downward tracing (directional code 4) have the highest dependency in determining the classes [3].

8. Conclusions

A method for calculating relative dependency between the input and output nodes of a given feedforward network has been proposed. The method is "closely coupled" between the two tasks of feature selection and classification. A number of experiments have been designed based on: 1) a set of string data for 2-D shape recognition using artificial and electromagnetic biomolecular image data, and 2) the parity-three function. Its usefulness for identifying irrelevant input node(s) has also been demonstrated.

9. References

[1] Chiu, David K. Y. (1993a). Transformation systems and neural networks. *Proceedings of World Congress on Neural Networks*, Portland, July 11-15, vol. III, pp.424–428,.

[2] Chiu, David K. Y. (1993b). Towards an event-space self-configurable neural network *Proceedings of IEEE International Conference on Neural Networks*, San Francisco, March 28-April 1, pp.956-961.

[3] Chiu, David K. Y.; Harauz, George; Greenspan, Neil S.; Roux, Kenneth H. (1994). Structured object representation of bimolecular images for two-dimensional shape discrimination via feedforward networks. *Journal of Computer-Assisted Microscopy, vol. 6, no. 3*

[4] Draper, N. R. (1986). *Applied regression analysis*. John Wiley, New York.

[5] Fu, Limin (1993). *Neural networks in computer intelligence*. McGraw Hill, New York.

[6] Goldfarb, Lev (1990).On the foundations of intelligent process. *Pattern Recognition* vol.23, no.6, pp.595-616.

[7] Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley, New York.

[8] Kittler, J. (1986). Feature selection and extraction. *Handbook of pattern recognition*, Academic Press, New York.

[9] Pao, Yoh-Han (1989). *Adaptive pattern recognition and neural networks*. Addison Wesley, New York.

[10] Ratkowsky, David A. (1983). *Nonlinear regression modelling*. Marcel Dekker, New York.