Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), *12*, 259–272. Printed in the USA.
Copyright © 1998 Cambridge University Press 0890-0604/98 $12.50

A binary classifier with applications to poorly defined engineering problems

IAN FLOOD[1] AND NABIL KARTAM[2]

[1]M.E. Rinker, Sr. School of Building Construction, University of Florida, Gainesville, FL 32611, U.S.A.
[2]Department of Civil Engineering, University of Kuwait, Safat, Kuwait.

(RECEIVED September 2, 1997; ACCEPTED February 19, 1998)

Abstract

The paper proposes and evaluates a binary network architecture and complementary training algorithm designed for pattern classification, with applications in a variety of engineering problems. The advantages of the system are that it can always converge on zero error for a set of unambiguous training patterns (if required), converges rapidly, and circumvents the issue of how many hidden neurons to incorporate in a network. The main benefit of the proposed system over Hamming networks, its main counterpart, is that it can group patterns into different classes along any boundary. The system is shown to outperform 1) the Hamming network for a character recognition problem where the images are subject to both position change and noise; and 2) a radial-Gaussian network in a truck-type classification problem. A variant of the system, where hidden neuron thresholds are set to zero, is shown to further improve performance if a comprehensive set of noise-free input patterns are available for training.

Keywords: Classifier; Hamming Networks; Noise Tolerance; Rapid Training; Truck Classification

1. INTRODUCTION

A necessary attribute for many classification devices (including associative memory) is the ability to operate within an indefinite and noisy environment. Hand-writing readers, for example, are often expected to recognize characters that vary in position, size, rotation, and style, and are subject to noise. Similarly, a classifier that recognizes truck types (essentially the axle configuration) from the strain it induces on a bridge girder should work irrespective of the spacings between the axles, the loads on each axle, and vibration noise in the structure. Moreover, to ensure utility, the classifier should be able to generalize within this environment, providing correct classifications of patterns not used for training. A wide variety of neural-based systems have been used as classifiers, including the backpropagation network (Rumelhart et al., 1986; Tsoi, 1990), radial-Gaussian networks (Poggio & Edelman, 1990; Gagarin et al., 1994), the Hopfield network (Hopfield, 1982; Hopfield et al., 1983; Hopfield, 1984), and the Brain-State-in-a-Box network (Anderson et al., 1977; Anderson, 1983).

An important type of classifier for binary input to output mapping is the Hamming network (Lippmann, 1987; Lippmann et al., 1987). Lippmann (1987) has shown the system to outperform the Hopfield network for character recognition in a noisy environment. Tsoi (1990) has further demonstrated the superior performance of Hamming networks for noisy problems, in a comparison with backpropagation and Hopfield models. The primary advantage of Hamming networks is simplicity both in their architecture and method of development. In particular, this means that they are quick to develop and modify. Such a characteristic is important in real-time engineering applications, such as in control systems, where new information about a problem is being generated all the time and needs to be assimilated in the model rapidly in order for it to be fully exploited. A major problem with Hamming networks, however, is that they cannot group patterns into different classes along arbitrary boundaries (see Section 3).This is unfortunate since many engineering problems, such as the classification of trucks from induced strain, involve a division of the problem domain along complicated boundaries.

The motivation for the development of the neural networking system presented in this paper was, therefore, to provide a device that demonstrated all the merits of Ham-

Reprint requests to: Nabil Kartam, Associate Professor, Department of Civil Engineering, Kuwait University, P.O. Box 5969, Safat, 13060 Kuwait.; E-mail: kartam@kuc01.kuniv.edu.kw

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

ming networks, but with none of their limitations, and so would be appropriate for real-time-operating engineering problems. In particular, the device was targeted at the problem of classifying trucks from the strain they induce in bridge girders, though need not be limited to this case. The end-system should be simple in architecture and rapid to train enabling modification on-the-fly, it should be tolerant of noise, and capable of dividing a problem domain into any arbitrarily complicated class regions.

The proposed system is a development of the Hamming network. It includes its own supervised training algorithm that is guaranteed to converge on zero error for a set of training patterns (provided that the data is not ambiguous), and determines as a matter of course the number of hidden neurons required to achieve zero training error.

2. NETWORK ARCHITECTURE AND TRAINING

2.1. Network structure and operation

The main part of the proposed network architecture, illustrated in Figure 1, comprises an input, output, and single hidden layer of neurons. A layered feedforward configuration is adopted with each output neuron connected to its own set of hidden neurons. The main network provides a direct mapping from a vector of binary inputs, $\boldsymbol{i}$, to a vector of binary outputs, $\boldsymbol{a}''$. A regulator neuron, feeding information back from the output neurons to the hidden neurons, is included to ensure that only a prescribed number of output neurons fire (usually one).

Binary values, i_x, presented at each of the input neurons are transmitted across the connections to the hidden neurons. Associated with each of these connections is a binary value $b_{x,n}$ which is applied to the transmitted value in an exclusive-or (XOR) operation . The results of this process over all connections to a hidden neuron are summed, providing the Hamming distance H_n between the input vector $\boldsymbol{i}$ and the template vector $\boldsymbol{b}_n$. Formally, this can be expressed as:

$$H_n = \sum_{x=1}^{X} (b_{x,n} XOR\ i_x). \tag{1}$$

If the value H_n is less than or equal to the threshold value T'_n (T'_n is a positive integer in the range 0 to X inclusive where X is the number of inputs to the network), the hidden neuron will fire by outputting binary 1, otherwise it will output binary 0. That is,

$$\begin{array}{ll} \text{IF:} & H_n \le T'_n \\ \text{THEN:} & a'_n = 1. \\ \text{IF:} & H_n > T'_n \\ \text{THEN:} & a'_n = 0. \end{array} \tag{2}$$

Each output neuron sums the values a'_n generated at the set of hidden neurons to which it is connected:

$$S''_y = \sum_{n=1}^{N} a'_n \tag{3}$$

and compares the result with a threshold value T''_y (T''_y can be any positive integer including zero). Firing of an output neuron works the opposite way round to that of the hidden neurons:

$$\begin{array}{ll} \text{IF:} & S''_y > T''_y \\ \text{THEN:} & a''_y = 1 \\ \text{IF:} & S''_y \le T''_y \\ \text{THEN:} & a''_y = 0. \end{array} \tag{4}$$

The regulator neuron sums the values generated by the output neurons:

$$S''' = \sum_{y=1}^{Y} a''_y \tag{5}$$

and compares the result with a threshold value T''' (where T''' is any integer between 1 and $X-1$). Firing of the regulator neuron operates as follows:

$$\begin{array}{ll} \text{IF:} & S''' < T''' \\ \text{THEN:} & a''' = -1. \\ \text{IF:} & S''' = T''' \\ \text{THEN:} & a''' = 0. \\ \text{IF:} & S''' > T''' \\ \text{THEN:} & a''' = 1. \end{array} \tag{6}$$

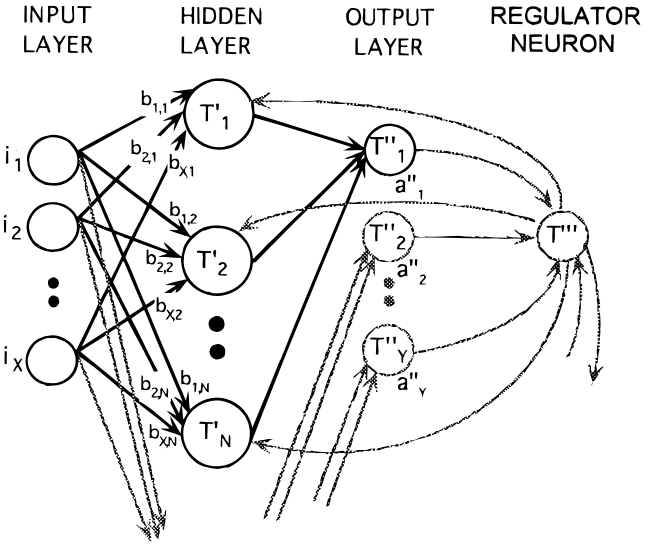


Fig. 1. Network architecture.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

The value generated at the regulator neuron is transmitted back to the hidden neurons where it is added to the previous summed input, modifying Eq. (1) so that

$$H_{n,j+1} = H_{n,j} + a_j'''. \tag{7}$$

The network operates in this recursive manner until the activation at the regulator neuron, a''', either becomes equal to 0 (at which point the number of output neurons firing will be equal to T''') or flips from -1 to $+1$.

2.2. Graphical interpretation of network operation

The architecture of these networks and the training algorithm described below are complementary in design. The rationale behind this design can readily be understood through a graphical interpretation of the operation of the system. In this view, the domain within which the network operates is considered to be a square, cube, or hyper-cube, with an order of dimensions equal to the number of input neurons. A network with two inputs, for example, would operate within a square input domain such as illustrated in Figure 2. Each dimension of the square ranges from 0 to 1 and corresponds to one of the input neurons in the network, while each corner of the square corresponds to one possible vector of inputs. If an input vector causes the output neuron to fire, the corresponding corner of the square is marked with a shaded circle. If, on the other hand, the output neuron does not fire the corner is marked with a clear circle.

Each hidden neuron (as with any simple perceptron) partitions the square with a line, marking a boundary between those input vectors that do and do not cause the hidden neuron to fire, as indicated in Figures 2c and 2d. The line is positioned relative to the corner of the square that matches the hidden neurons template vector $\boldsymbol{b}_n$, at a distance given by the hidden neuron's threshold T_n'. If, for example, $T_n' = 1$ (as is the case for both hidden neurons shown in the figure), the line will be positioned just beyond a Hamming distance $H_n = 1$ from $\boldsymbol{b}_n$. Note that the input vectors that cause the hidden neuron to fire are those falling on the $\boldsymbol{b}_n$ side (the firing side) of the line.

For an output neuron to fire, there must be more than T_y'' hidden neurons firing. If $T_y'' = 1$, for example, then an input vector would have to fall on the firing side of the partitioning line of at least two hidden neurons to cause the output neuron to fire. For the network shown in Figure 2a, an input vector must be positioned where the firing regions of two hidden neurons overlap. This is indicated by the shaded region in Figure 2b.

Given an appropriate output threshold and set of hidden neurons, it is possible to provide any set of binary input to output mappings within this type of network. A trivial solution would be to use a separate hidden neuron to partition each corner of the cube where an output of 1 is required, setting $T_n' = 0$ in each case, and $T_y'' = 0$. In this way, every problem requiring an output of 1 would be satisfied by its own hidden neuron. However, the training algorithm described below, while not guaranteed to find the minimum number of hidden neurons required to solve a problem, will

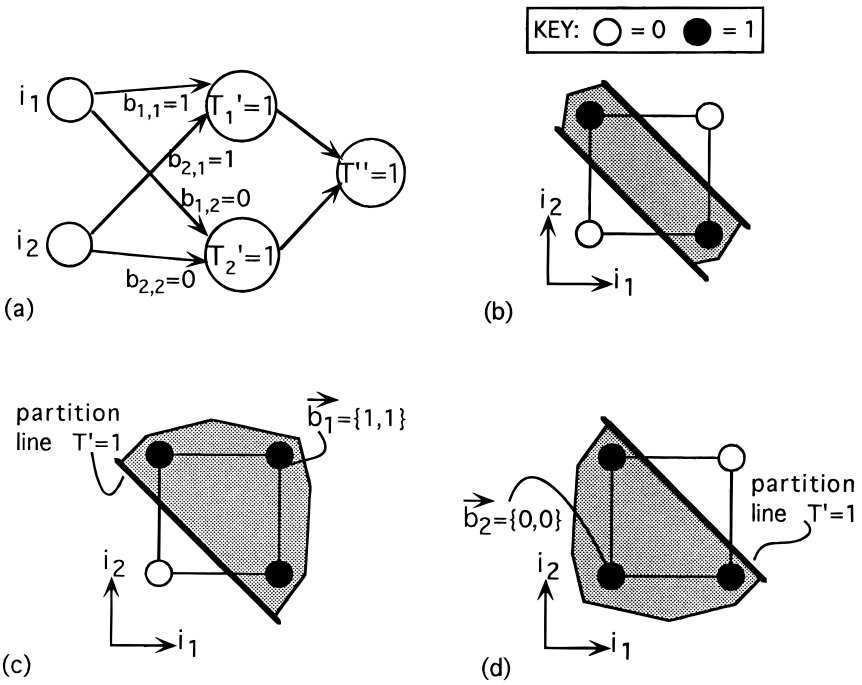


Fig. 2. Graphical interpretation of network operation. (a) Two hidden neuron networks. (b) Region of the problem domain that causes the output neuron to fire. (c) Firing region of first hidden neuron. (d) Firing region of second hidden neuron.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

typically provide a considerably more efficient solution than this, using many fewer hidden neurons than problems requiring an output of 1.

For classification problems, it is usually required that an input vector causes just one output neuron to fire, that representing the class to which the input vector belongs. The required number of output neurons that should fire can, however, be greater than one and is defined by the regulator neuron's threshold T'''. When insufficient training patterns are available to define the precise boundaries of each class, or when there are errors in the input vector to be classified, it is possible that too few or too many output neurons will fire. The regulator neuron attempts to correct such situations by either increasing or decreasing the sensitivity of the hidden neurons, as appropriate, which in turn increases or decreases the likelihood of firing of the output neurons.

Consider, for example, a situation where an input vector causes too few output neurons to fire. In this case, the regulator neuron will output -1 thus reducing the summed input to the hidden neurons by 1. This, in effect, moves the partition lines for the hidden neurons outwards by a Hamming distance of 1, extending their firing regions. Correspondingly, the firing regions of the output neurons will be extended by a Hamming distance of 1, increasing their likelihood of firing in response to the current input vector. The firing regions are extended as such until sufficient output neurons are firing. Ultimately, the output neurons that fire will be those with the closest original firing regions to the input vector. If too many output neurons are firing then the opposite occurs, with the firing regions of the hidden and output neurons being decreased in increments of 1 until the required number of outputs are firing. Either way, the maximum number of iterations required to achieve the required balance in the number of output neurons firing will not exceed the number of inputs to the network.

2.3. Training algorithm

The training algorithm falls into the supervised class, making use of a set of P training patterns each comprising an input vector $\boldsymbol{i}_p$ and corresponding target output vector $\boldsymbol{t}_p$. Training operates on the main section of the network, exclusive of the regulator neuron. Since each output neuron has its own set of hidden neurons, training can be more conveniently described for the one output neuron case. If there are two or more outputs, the training for each is performed in parallel but is otherwise the same as in the single output neuron case.

Training proceeds by developing one hidden neuron at a time. Each successive hidden neuron is trained on the component of the problem its predecessors failed to learn, that is, the remaining error. Hidden neurons are added as such until the remaining error is zero. For a network comprising one output neuron, the procedure is as follows:

STEP 1: *Initialize the network.* The network is first set up with the required number of input neurons, an output neuron, but with no hidden neurons. The threshold T'' on the output neuron is set to 0. For the network to output the correct result for the pth training pattern, the value generated for S_p'' should be greater than the threshold T'' if the target output is 1, or it should be less than or equal to T'' if the target output is 0. An error vector $\boldsymbol{\epsilon}$ is established to measure the distance from these objectives for each training pattern:

$$\epsilon_p = t_p + T'' - S_p''. \tag{8}$$

The objectives then become

$$\epsilon_p \leq 0 \qquad \text{if } t_p = 1$$

and

$$\epsilon_p \geq 0 \qquad \text{if } t_p = 0.$$

Since the network starts with no hidden neurons and $T'' = 0$, each element of the error vector is initialized to $\epsilon_p = t_p$. The total error in the network is measured as

$$E = \sum_{p \subset \Pi_\epsilon} |\epsilon_p|, \tag{9}$$

where Π_ϵ is the set of all training patterns for which the network produces an incorrect output.

STEP 2: *Determine whether most errors are for training patterns with a target of 1 or a target of 0.* A count is made of the number of training patterns P_{high} in the set Π_{high} (comprising all training patterns that have a target of 1 but the network produces 0) and the number of training patterns P_{low} in the set Π_{low} (comprising all training patterns that have a target output of 0 but the network produces a 1). This is undertaken by checking how many of the elements in the error vector $\boldsymbol{\epsilon}$ do not satisfy the $\epsilon_p \leq 0$ and $\epsilon_p \geq 0$ requirements respectively (see Step 1). A hidden neuron is then added to the network, as described in the following steps, to reduce the number of patterns in either Π_{high} or Π_{low}, whichever is the largest set.

STEP 3: *Add a hidden neuron to the network and establish the values of its template vector* $\boldsymbol{b}_n$. If the number of training patterns in Π_{high} is greater than or equal to the number in Π_{low}, then follow Step 3a otherwise follow Step 3b.

STEP 3a: The center of gravity $\boldsymbol{\gamma}$ (measured in the input domain) of all training patterns in the set Π_{high} is determined. Each element of the vector $\boldsymbol{\gamma}$ is calculated simply as

$$\gamma_x = \left(\sum_{p \subset \Pi_{high}} i_{p,x} \right) \Big/ P_{high}. \tag{10}$$

The training patterns in set Π_{high} are then searched to determine which is nearest to the center of gravity $\boldsymbol{\gamma}$. The nearest training pattern $\hat{p}$ is that where $\sum_{x=1}^{X} |i_{p,x} - \gamma_x|$ is

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

smallest. A hidden neuron is added to the network and its template vector is set equal to the input vector of training pattern $\hat{p}$, that is $\boldsymbol{b}_n = \boldsymbol{i}_{\hat{p}}$. The rationale behind this step is that the center of gravity will indicate the corner of the input space towards which the erroneous training patterns are clustered, and thus focusing the hidden neuron at a corner near this point facilitates the removal of a significant number of errors.

OR

STEP 3b: The parameters $\boldsymbol{\gamma}$ and $\hat{p}$ are determined as described in Step 3a except that P_{low} and Π_{low} are used in place of P_{high} and Π_{high}. A hidden neuron is added to the network and its template vector is set equal to the logical NOT of the input vector of training pattern $\hat{p}$, that is

$$\boldsymbol{b}_n = NOT\,(\boldsymbol{i}_{\hat{p}}). \tag{11}$$

Following this, a value of 1 is added to the output threshold T'' and, so that Eq. (8) is maintained, a value of 1 is added to each component of the error vector $\boldsymbol{\epsilon}$. The rationale here is essentially the same as that of Step 3a. In this case, however, the erroneous training patterns are causing the output neuron to fire when it should not—the opposite of the situation in Step 3a. The hidden neuron, therefore, needs to be set up in a way that suppresses (rather than promotes) the possibility of the output neuron firing when one of the erroneous training patterns are presented to the network. This is achieved by focusing the hidden neuron on the opposite corner of the input space to that where the erroneous training patterns are clustered [see Eq. (11)] and incrementing the output threshold as described.

STEP 4: *Determine an optimum value for the hidden neuron's threshold* T'_n. All possible values for T'_n (from 0 to X inclusive) are checked to see which will provide the greatest reduction in total error E in the network [see Eq. (9)]. This can be achieved with just one pass through the training patterns. Each training pattern is examined to see if its contribution to the total error would be reduced or increased by the addition of the hidden neuron. If the pth training pattern's target output is 1 and $\epsilon_p > 0$, then its contribution to the total error E will be reduced if the threshold T'_n is set at least equal to the following Hamming distance:

$$H_{p,n} = \sum_{x=1}^{X} (b_{x,n} XOR i_{p,x}).$$

A score of 1 is therefore added to a variable, $\Delta_{k=H_{b,n}}$, associated with this Hamming distance. Conversely, if the pth training pattern's target output is 0 and $\epsilon_p \leq 0$, then its contribution to the total error will increase if the threshold T'_n is set at least equal to the Hamming distance $H_{p,n}$. In this case, a score of -1 is added to the variable $\Delta_{k=H_{b,n}}$. The cumulative scores for all possible Hamming distances are then calculated:

$$C_{m=0\ldots X} = \sum_{k=0}^{m} \Delta_k.$$

Each cumulative score C_m provides a measure of the reduction in error that can be achieved by setting the threshold T'_n to a Hamming distance of m. The value of m at which the highest positive cumulative score is achieved is therefore the value selected for the threshold T_n. If there are two or more optimum threshold values the choice between them is arbitrary, though in this paper the smallest value is adopted.

STEP 5: *Adjust the error vector.* The values of the error vector $\boldsymbol{\epsilon}$ are adjusted to take account of the contribution of the hidden neuron. This requires each error ϵ_p to be reduced by a value of 1 if the training pattern causes the hidden neuron to fire, that is,

$$\epsilon_p^{new} = \epsilon_p^{old} - a'_{p,n}.$$

Further hidden neurons are added to the network by repeating Steps 2–5 inclusive until the total error E reaches 0.

3. COMPARISON WITH HAMMING NETWORKS

The proposed system has a superficial similarity to several other constructive networking systems (such as the cascaded-correlation training architecture [Fahlman & Lebiere (1990)] in that network development proceeds by developing one hidden neuron at a time. However, the proposed system is very distinct from such systems both architecturally and in the method of network development. On the architectural side, the proposed system uses binary masks rather than real-valued weights on connections, and specifically measures Hamming distances at the point of integration on the hidden neurons. In this sense, it is much more akin to the Hamming networks of Lippmann et al. (1987). As far as training is concerned, the binary masks in the proposed system are determined directly from the binary patterns of the training pattern set (rather than by means of an error gradient-descent search algorithm) again placing it close to Hamming networks and distinct from other constructivist systems. Finally, the area of application of the system is binary pattern classification as with Hamming networks.

For these reasons, the Hamming network is viewed as the main counterpart to the proposed system, and was therefore selected for comparison. This section illustrates the similarities and differences between the two systems in terms of their architecture, method of development, and scope of application. Following this, a quantitative evaluation of the relative performances of the two systems is made for two classification problems.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

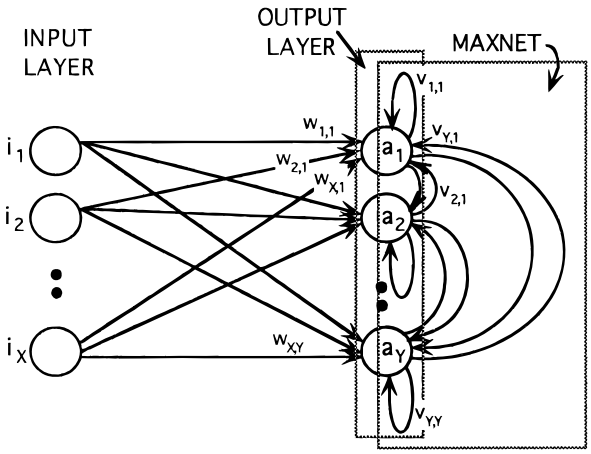


Fig. 3. Hamming network.

Figure 3 shows a schematic of a Hamming network. The first stage in this network comprises an array of perceptron-like neurons, each of which measures X minus the Hamming distance H_y between the input vector $\boldsymbol{i}$ and a template vector $\boldsymbol{b}_y$:

$$a_y = X - H_y$$

$$= X - \left[\sum_{x=1}^{X}(b_{x,y}) - \sum_{x=1}^{X}(w_{x,y}\, i_x)\right],$$

where

$$w_{x,y} = 1 \qquad \text{if } b_{x,y} = 1,$$

$$w_{x,y} = -1 \qquad \text{if } b_{x,y} = 0.$$

The second stage of the network, termed the maxnet, determines which of the output neurons in the first stage has the largest activation a_y (note that, the output with the largest activation is that with the smallest Hamming distance between its template vector $\boldsymbol{b}_y$ and the input vector $\boldsymbol{i}$). The maxnet operates in an iterative manner, forcing the activation of all output neurons to equal zero, except that which has the largest initial value. The new activations after each iteration are

$$a_{y,t+1} = f\left[\sum_{z=1}^{Y}(\nu_{y,z}\, a_{y,t})\right],$$

where

$$f(\tau) = \tau \qquad \text{if } \tau > 0,$$

$$f(\tau) = 0 \qquad \text{if } \tau \leq 0,$$

also

$$\nu_{y,z} = 1 \qquad \text{if } y = z,$$

$$\nu_{y,z} = \nu \qquad \text{if } y \neq z, \qquad \text{where } -1/(X-1) < \nu < 0.$$

The main similarity between the system proposed in this paper and the Hamming network system is that both compute Hamming distances from templates as part of the classification process. However, Hamming networks assign just one template to each class and base their classification on the distance between this and the input vector. This severely limits the way in which patterns can be grouped into different classes. Networks of the type proposed in this paper, on the other hand, can assign any number of templates to each class and base their classification on whether the input vector is sufficiently close (in Hamming distance) to a critical number of these templates—the critical number of templates can be anything from 1 to N (the number of templates assigned to the class) and the acceptable Hamming distances can vary from template to template. This allows the network to differentiate classes along any boundary. The extra flexibility afforded by this system makes it potentially better suited to, for example, pattern recognition problems that involve lateral shift, rotation and size variance.

4. PERFORMANCE RESULTS

4.1. Character recognition

The first performance test was an assessment of the network's ability to classify images of the ten numeric characters "0" to "9", subject to random positional shift as well as noise. This problem was selected to illustrate the method of application and assess the relative performance of the proposed system. Each character was designed to fit within a 7×11 pixel grid which in turn could be located anywhere within a 10×16 pixel grid. Each pixel could assume one of two shades, black or white, representing binary 1 and 0, respectively. The ten characters, located at random positions but with no noise, are illustrated in Figure 4a. In the experiments reported here, noise was introduced to the images by passing each pixel through a discrete, noisy, memoryless channel, as described in Lippmann et al. (1987). A symmetric channel was adopted, $p\langle 0|1\rangle = p\langle 1|0\rangle = \rho$ (i.e., the probability of a 0 being changed to a 1 was equal to the probability of a 1 being changed to a 0)

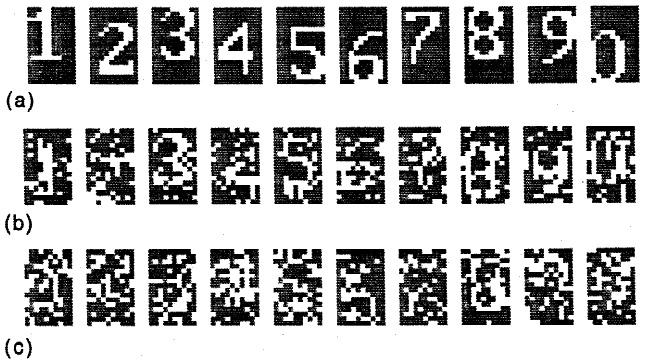


Fig. 4. Example noisy and positionally shifted numeric characters. (a) Characters with 0% noise. (b) Characters with 15% noise. (c) Characters with 25% noise.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

and the range of noise considered was $0\% \leq \rho \leq 50\%$ (50% giving maximum entropy). Examples of the ten characters, randomly positioned, with 15% and 25% noise are shown Figures 4b and 4c, respectively.

Eleven sets of training patterns were established, the first with ρ set to 0% for all example images, the second with ρ ranged from 0% to 5%, and so on increasing the range by 5% up to the 11th set which had ρ ranged from 0% to 50%. In each case, the amount of noise added to an example image was selected using a random value taken with uniform probability within the prescribed range for the set. In the third training set, for example, each occasion an image was generated, the value for ρ was set equal to a random value between 0% and 10%. The position of each example image within the 10×16 pixel grid was also selected randomly with uniform probability distribution. Each set of training patterns had 10,000 example images, 1000 of each type of character.

In addition, a set of 11 test pattern sets were produced, in a similar way to the training patterns except that ρ was fixed in each set. For example, in the second test pattern set, ρ was set to 5% for all example images rather than being ranged from 0% to 5%. A total of 1000 test patterns comprising 100 examples of each type of character were generated for each of the 11 sets.

Eleven networks were trained, one per training pattern set, and tested on all 11 test pattern sets. Figure 5 plots the performance of each network (measured as percentage of the test patterns correctly classified) against the percentage noise in the test patterns. The best performer, for all levels of noise in the test patterns, was the network trained with a 0–10% noise range. This network correctly classified 94% of patterns at 15% test pattern noise, reducing to 82% correct classifications at 20% test pattern noise.

A plot of the percentage of correctly classified patterns, containing 25% noise, against the range of noise used for training, is provided in Figure 6. From this, it can be seen that performance improves rapidly up to the 0–10% range after which it declines steadily. The reason for the decline is that, beyond the 0–10% range, sufficient noise is being generated to result in training patterns that are ambiguous (i.e., falling relatively close in Hamming distance to other patterns within the input domain). These patterns are learned by the network, but, due to their ambiguous placement, cause misclassification of test patterns.

Figure 7 plots the performance of the best network (that trained on 0–10% noise) against the percentage noise in the test patterns, for each of the ten characters. From this it can be seen that the network's worst performance is for characters "3" and "8", possibly due in part to their similarity in form and thus confusion. The best performance is for character "1" which, even at 50% test pattern noise, was correctly classified more than 50% of the time. The reason for this is that the network, at this level of noise, was classify-

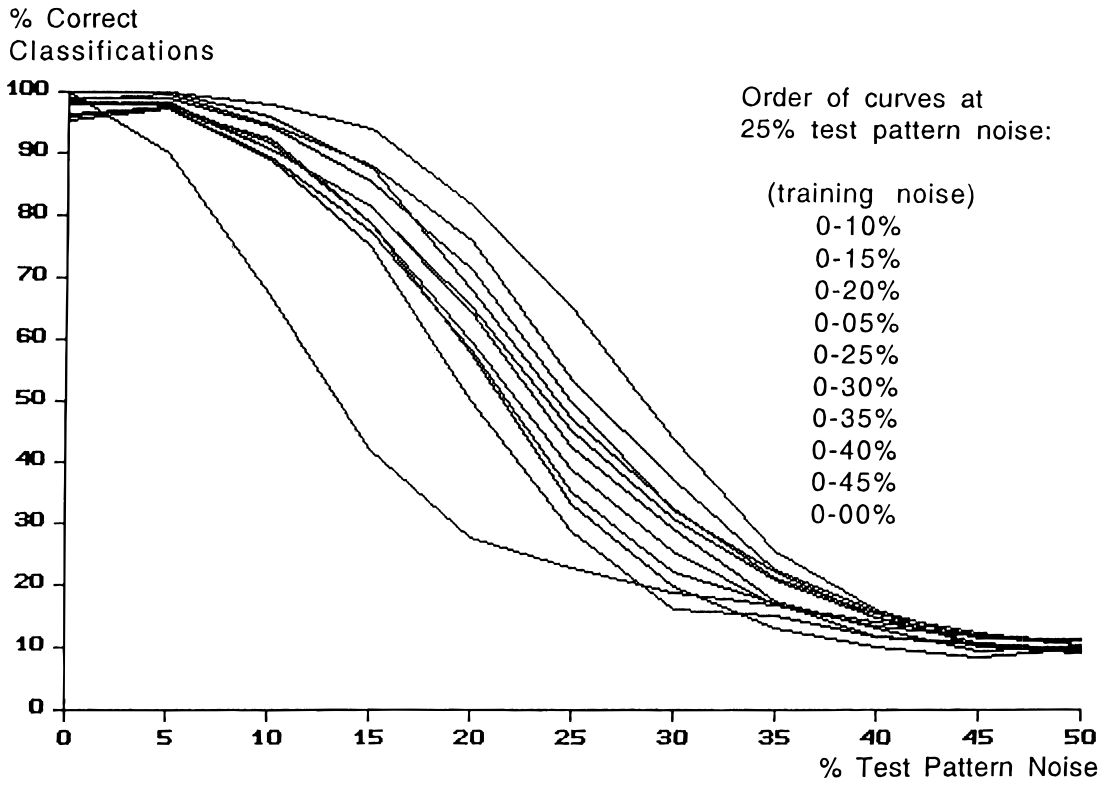


Fig. 5. Effect of test pattern noise on network performance.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

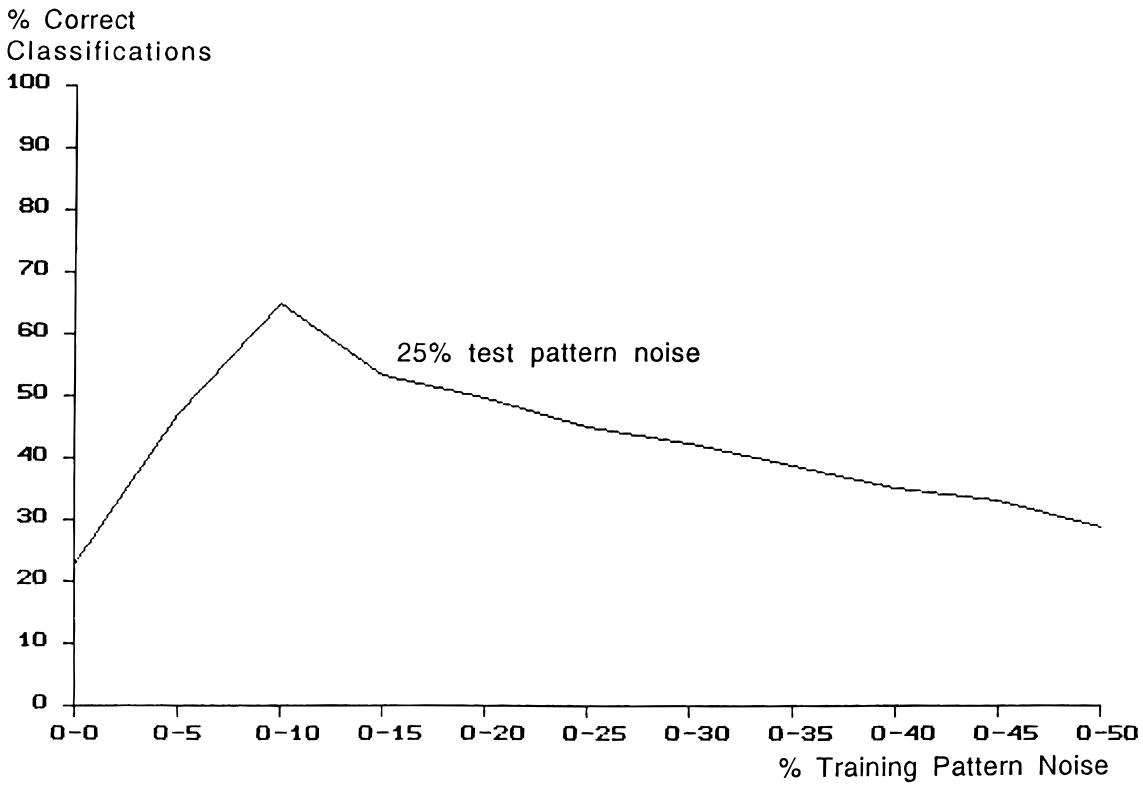


Fig. 6. Effect of training noise on the ability of a network to classify a set of test characters.

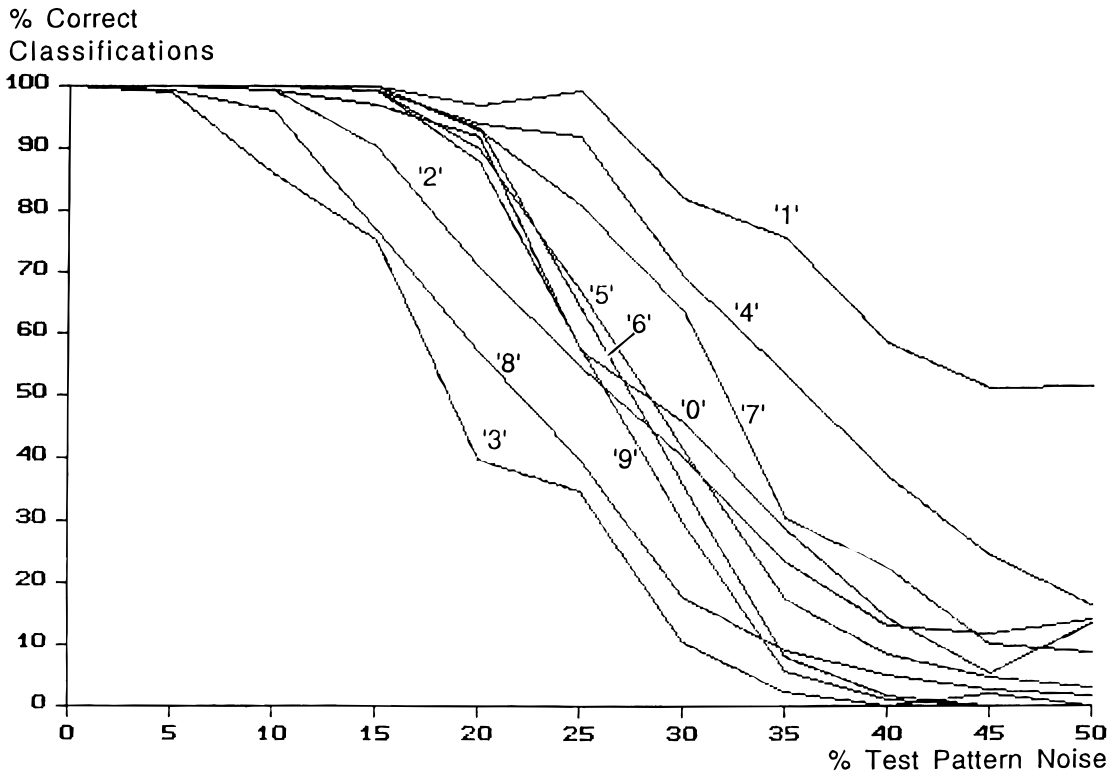


Fig. 7. Network performance for individual characters.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

ing patterns as the character "1" with a very high frequency thus increasing its chances of making this classification correctly. Indeed, the average percentage of correct classifications for all characters was a little over 10% at the 50% noise limit—this is close to the expected 10% value at maximum entropy for a ten character set. A wide range in performances between characters, as displayed here, may be undesirable. If this is the case, a better solution may be achieved by increasing the number of training patterns of the poorly performing characters and the characters with which they tend to be confused.

4.2. Comparison with the zero-threshold variant and Hamming networks

The performance of the best network was compared to that of a Hamming network developed for the same problem. The Hamming network was set up using, for templates, each of the characters located as near as possible to the center of the 10×16 pixel grids. Its performance was assessed using the 11 sets of test patterns, the results of which are plotted as curve B in Figure 8. Also shown for comparison, as curve A, is the best network from the previous set of experiments. The Hamming network's performance is generally very poor. This can be attributed, at least in part, to the fact that the characters were allowed to shift vertically and horizontally within the grid. Interestingly, though, at 45% test pattern noise, the Hamming network showed a marginally better performance than all other networks.

An assessment was also made of the performance of a variation of the proposed system, where the hidden neuron thresholds T_n' are all set to zero during training (though are allowed to vary during application). By setting the thresholds to zero, the network is forced to develop one hidden neuron for each training pattern. In this sense, the zero-threshold network is very similar to the Hamming network. The main difference, however, is that it is not limited to one template for each class. The network was first developed using the set of training patterns with 0% noise. This meant that 24 hidden neurons were developed for each output, one for each possible position of a character within the 10×16 pixel grid. The performance of this network is plotted as curve C in Figure 8. Clearly, this zero-threshold network outperforms all other models, with 96% correct classifications at 25% test noise.

For the above problem, it is apparent that there are 24 archetypal noise-free templates for each class (one for each character position within the 10×16 grid). For many problems,

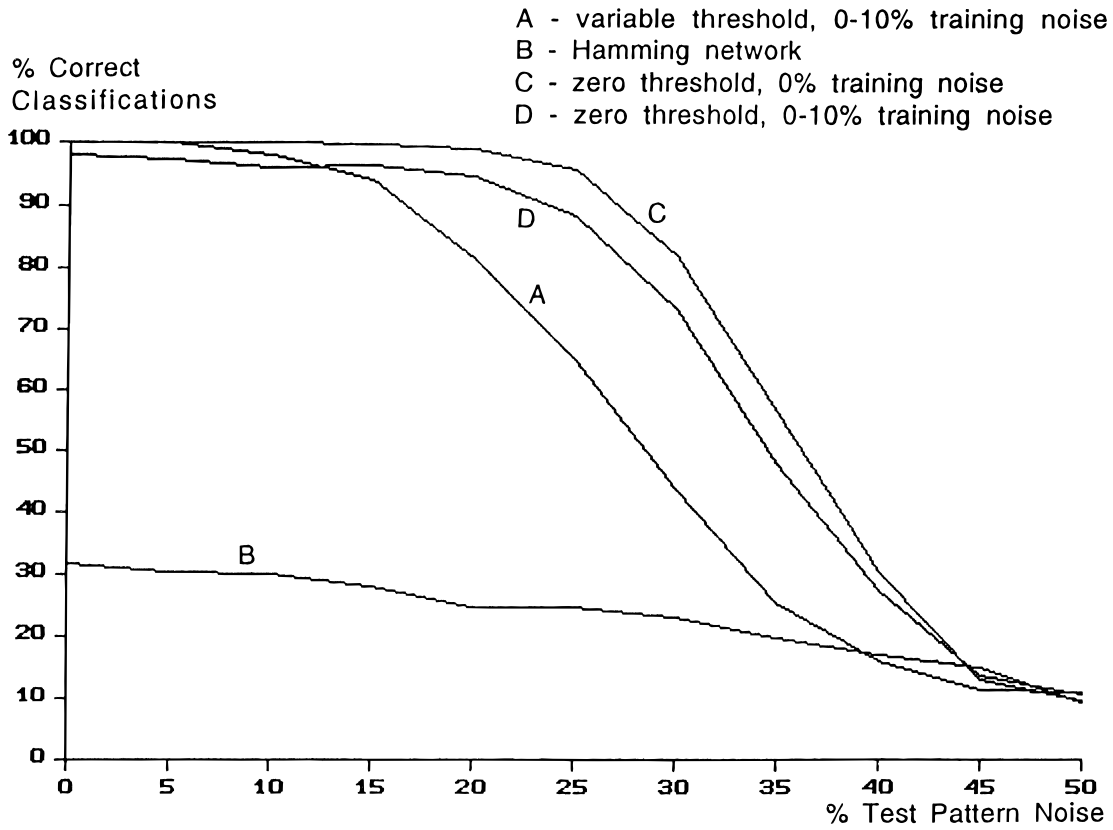


Fig. 8. Comparison of the performances of the proposed system, a variation of this system with fixed hidden neuron thresholds, and a Hamming network.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

however, it is not possible to define a comprehensive set of idealized templates. There is no singularly correct set of templates for handwritten characters, for example. It is also difficult in such cases to distinguish noise from the intended pattern. It was decided, therefore, to develop the zero-threshold network using the set of training patterns comprising 0–10% noise, to see how uncertainty can affect its performance. The results of this experiment are plotted in Figure 8 as curve D. This version of the zero-threshold network demonstrated good performance, though was generally below that of the zero-threshold network developed with the noise-free patterns. Also, until somewhere between 10% and 15% test pattern noise, its performance was below that of the network with variable thresholds. In summary, as far as correct classification of patterns is concerned, if a set of idealized templates are available, the zero-threshold variant appears to be the best approach. If, on the other hand, an idealized set of templates are not available then the variable-threshold approach appears better for lower noise levels and the zero-threshold approach for higher noise levels. However, these observations, in particular the latter, should not be taken as general principles. Further studies are required covering a variety of problems, including those that are poorly defined such as handwritten character recognition.

4.3. Network complexity

A direct means of comparing the complexity of the different types of network is in terms of the number of templates they comprise. For the proposed networking system, the number of templates is equal to the total number of hidden neurons, whereas, for a Hamming network, the number of templates is equal to the number of output neurons.

Figure 9 plots, for the character recognition problem, the number of hidden neurons (templates) required in the variable-threshold network to reduce the total error to zero, for each of the 11 training pattern sets. The best of these networks (that trained on 0–10% noise) incorporated 923 templates. Of the other approaches considered, the Hamming network incorporated 10 templates, the zero-threshold network developed with 0% noise included 240 templates, and the zero-threshold network developed with 0–10% noise included 10,000 templates. While the Hamming network was by far the smallest, this was of little value in view of its poor performance for this set of experiments. Also, while the zero-threshold network developed with 0% noise was the second smallest (about one quarter the size of the best variable threshold network), this might not hold for problems with more elements in the input patterns or that have more variance (such as occurs with rotation and size distortion of characters).

It is possible that many of the hidden neurons developed in a network were largely redundant, making little or no contribution in the classification of test patterns. An experiment was performed, therefore, to assess the effect on performance of varying the number of hidden neurons in the variable threshold network trained on 0–10% noise. The strategy adopted was to range the maximum number of hidden neurons connected to each output. Nine limits were considered starting at 1 then 2, 4, 8, and so on doubling to an upper limit of 512 hidden neurons maximum for each output. It should be remembered that all training patterns may be learned for an output before the hidden neuron limit is reached. The results of these experiments are plotted in Figure 10. Clearly, for the cases considered, there was little improvement in the performance of the network moving beyond the 64 hidden neuron limit. Given this, the network could

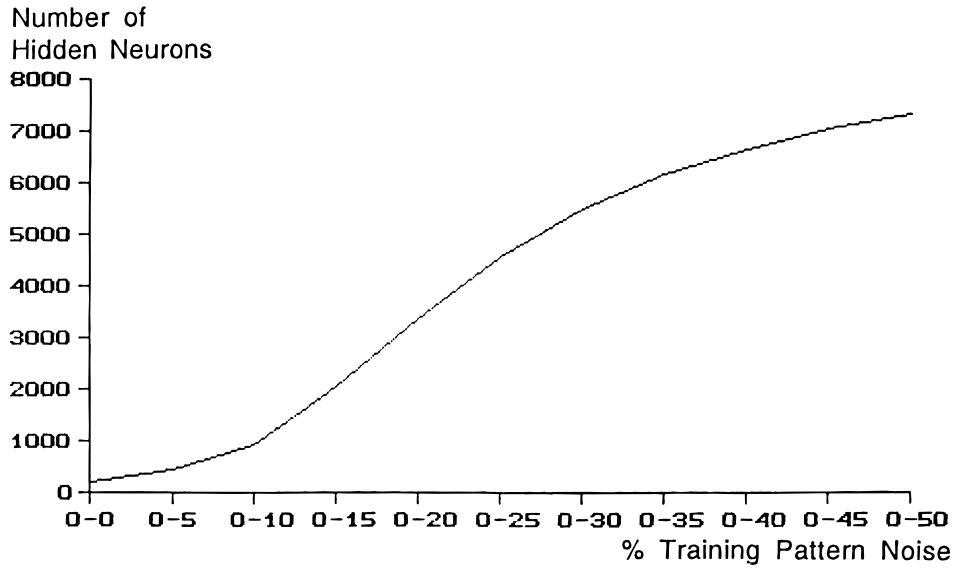


Fig. 9. Influence of training noise on the number of hidden neurons required to reduce the training pattern error to zero.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

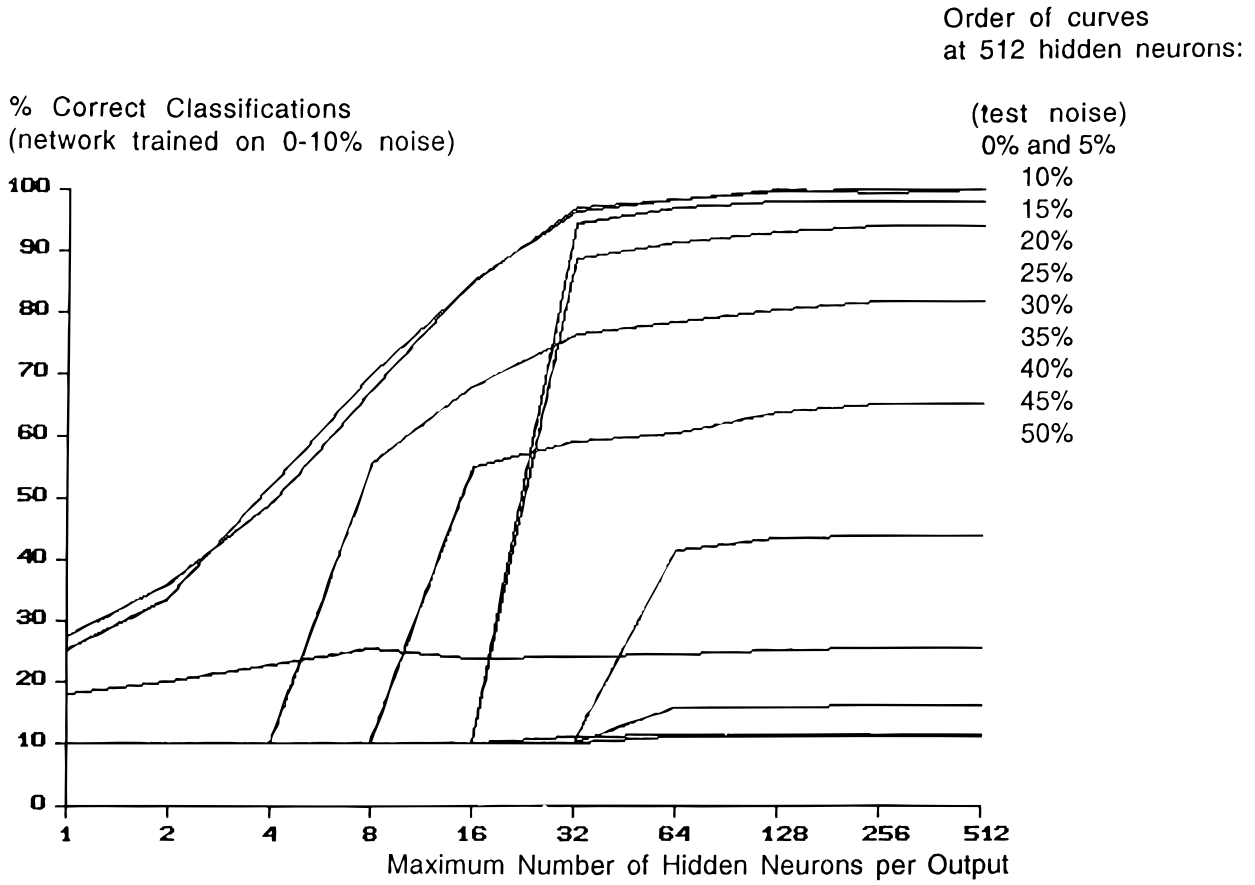


Fig. 10. Influence on performance of varying the maximum number of hidden neurons attached to each output.

have been reduced in size from 923 to 476 hidden neurons in total without seriously affecting its performance.

4.4. Truck-type classification problem

As a final experiment, the ability of the proposed neural networking system to recognizing the type of truck crossing a bridge was evaluated. Figure 11 shows the basic operation of this application. As a truck crosses a bridge, it induces strain in the girders. This is measured over the entire period of the truck crossing by a transducer, providing a strain–time curve such as that indicated in the figure. Each truck generates a characteristic strain–time curve, the form of which is a function of the velocity of the truck, the number

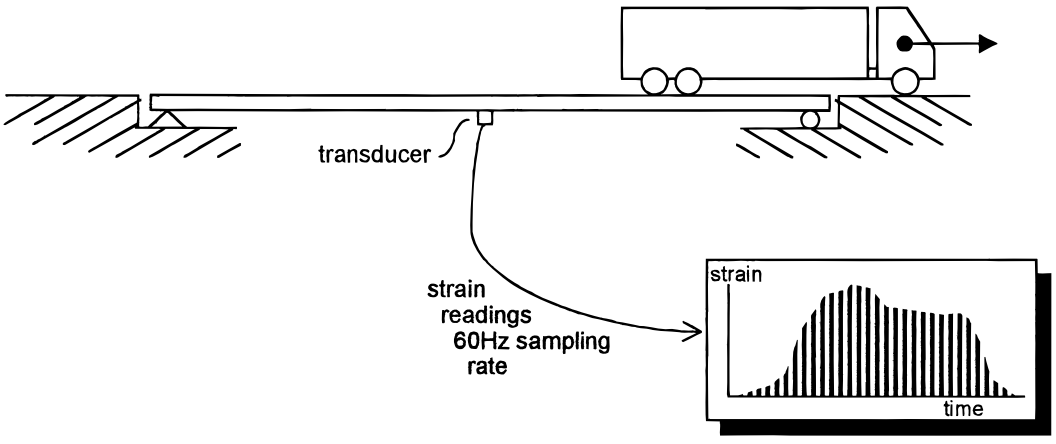


Fig. 11. Strain–time curve induced in bridge girder by the passage of a truck.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

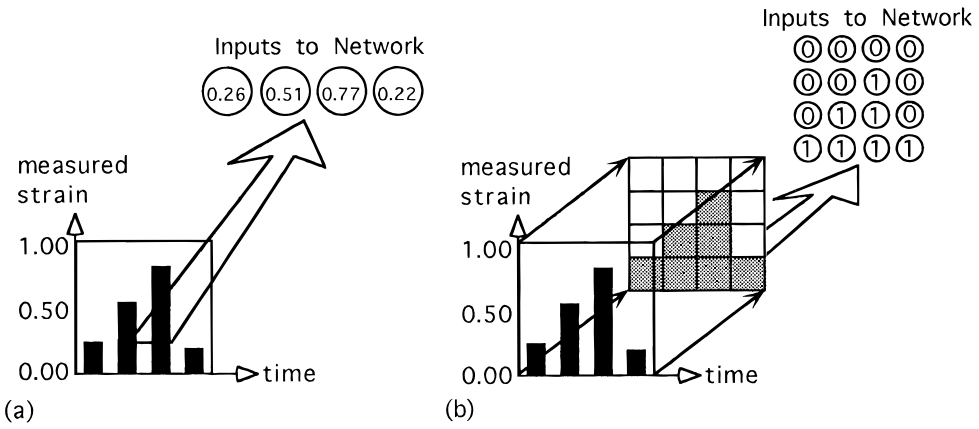


Fig. 12. Alternative methods of formatting strain–time curve for input to neural network. (a) Vector of real values. (b) Matrix of binary values.

of axles, the spacing between axles, and the load on each axle. The objective is to recognize the type of truck (according to the Federal Highways Administration system of classification) from its strain–time curve.

The solution to the problem involves presenting the neural network with data describing the strain–time curve induced by a truck. The response output from the network should be a correct classification of the truck type that resulted in the strain–time curve.

For comparison, the performance of the proposed neural system was compared to that of a radial-Gaussian based neural network (Flood, 1991) previously shown to outperform a backpropagation network for the truck-type classification problem (Gagarin et al., 1992, 1994). The radial-Gaussian network considered by Gagarin et al. (1994) comprised an array of inputs, each receiving one of the strain readings in the strain–time curve, as indicated in Figure 12a. For the proposed binary networking system, the strain–time curve was mapped onto a matrix of binary values in the manner indicated in Figure 12b.

This study considered the bridge configuration adopted by Gagarin et al. (1994), a simply supported 17.1-m bridge span representing the westbound lane crossing the Bull Run on Interstate 66 near Washington DC.

Training patterns were established, as with Gagarin et al., for the nine classes of truck most common in traffic (see Figure 13), with the range in axle spacings and axle loads listed in Table 1. Three values (the minimum, maximum, and central value) were considered for each axle spacing and axle load, for each truck class, giving the number of alternative truck configurations listed in the end column. Each truck configuration was run at a speed of 80.1 km/h, 96 km/h, and 112 km/h, across a single girder model of the bridge, and strain readings were generated based on a 60-Hz sampling rate, providing a total of 4131 strain–time curves. Each curve covered the duration of the truck crossing event, and was divided into 32 equal groups of strain readings—the values within each group were then averaged. Each resultant strain–time curve was used to establish a training pattern. For the radial-Gaussian network, each strain–time curve provided directly a 32-element input vector of a training pattern (see Figure 12a). For the proposed binary network, the series of strain readings were projected onto a 32×32 element grid to provide the binary matrix of input values for the training pattern (see Figure 12b). The projection was scaled so that a normalized strain reading of 1.0 would reach the top of the 32-element high grid. For both types of network, the output vector of each training pattern comprised a vector of binary values (one element for each type of truck). All elements in the output vector were set to 0, except that representing the class of truck that induced the strain–time curve which was set to 1.

A total of 1000 testing patterns were set up for each type of truck using axle spacings and loads selected at random from within the ranges given in Table 1, and with velocities between 80.1 km/h and 112 km/h.

The proposed binary networking system required a maximum of 917 hidden neurons to learn all the training patterns at an output, while the radial-Gaussian network was found to improve little beyond 1000 hidden neurons for any output. Table 2 shows the performance of both the proposed network and the radial-Gaussian network for the 9000 test patterns (1000 patterns per truck type). It is clear from these results that there is little to distinguish between the two types of network in terms of performance. However,

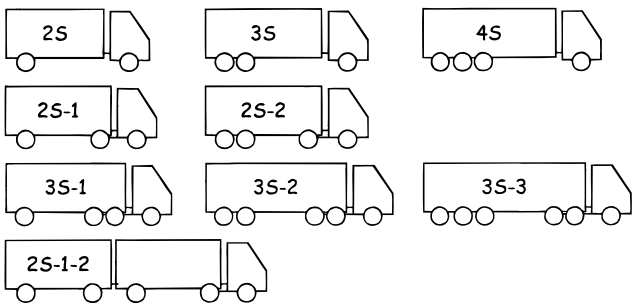


Fig. 13. Axle configurations of FHWA alternative classes of truck.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

Table 1. *Range of parameter settings used for establishing training patterns*

Type of Truck	Axle Loads (kN)						Axle Spacings (m)					No. of Training Patterns
	1	2	3	4	5	6	1–2	2–3	3–4	4–5	5–6	
2S	13.3–53.4	8.8–80.1	—	—	—	—	2.74–6.10	—	—	—	—	27
3S	13.3–53.4	8.8–80.1	8.8–80.1	—	—	—	2.74–6.10	1.22	—	—	—	27
4S	13.3–53.4	8.8–80.1	8.8–80.1	8.8–80.1	—	—	2.74–5.49	1.22	1.22	—	—	27
2S-1	13.3–53.4	8.8–80.1	8.8–80.1	—	—	—	2.74–4.88	5.49–1.6	—	—	—	243
2S-2	13.3–53.4	8.8–80.1	8.8–80.1	8.8–80.1	—	—	2.74–5.49	6.10–11.6	1.22	—	—	243
3S-1	13.3–62.3	8.8–71.2	8.8–71.2	8.8–80.1	—	—	2.74–6.10	1.22	6.10–11.6	—	—	243
3S-2	13.3–53.4	8.8–71.2	8.8–71.2	8.8–80.1	8.8–80.1	—	2.74–6.10	1.22	6.10–11.6	1.22	—	243
3S-3	13.3–53.4	8.8–71.2	8.8–71.2	8.8–80.1	8.8–80.1	8.8–80.1	2.74–6.10	1.22	6.10–11.6	1.22	1.22	243
2S-1-2	13.3–53.4	8.8–80.1	8.8–80.1	8.8–80.1	8.8–80.1	—	2.74–5.49	5.49	3.05	5.49	—	81

the proposed binary network has the advantage of being simpler in form and two to three orders of magnitude faster to train (see Section 4.5 below). While both networks performed reasonably well, there is potential for improvement by structuring the solution to work in stages. Specifically, most of the misclassifications were between trucks with similar axle configurations. For example, trucks of the 3S-1, 3S-2, and 3S-3 classes were often confused. Consequently, it may be possible to improve performance by having a primary network that simply identifies these general classes, followed by secondary networks (one for each general class) trained specifically to identify the type of truck within a given general class. This way, the tasks to be learned by the secondary networks will be greatly simplified, thus reducing the likelihood of confusion in classification.

4.5. Training speed

A major advantage of the proposed binary neural network system is the speed with which it can develop a network. For example, 0.058 s was the average processing time required to develop one hidden neuron in a network comprising 10 input neurons and 1 output neuron, using a set of 1000 randomly generated training patterns.[a] In contrast, the supervised training algorithm proposed by Flood (1991) for radial-Gaussian networks (adopted in the above problem) was about 431 times slower, taking an average of 25 s to train each hidden neuron for the same set of randomly generated patterns. The significance of this is that the radial-Gaussian training algorithm has in turn been shown, over a range of problems, to operate several orders of magnitude faster than the popular Generalized Delta Rule training algorithm of backpropagation networks (Flood, 1991; Gagarin et al., 1992).

Table 2. *Performance of neural networks for truck classification problem*

FHWA Truck Class	% Correct Classifications	
	Proposed Binary Network	Radial-Gaussian Network
2S	92.3%	93.1%
3S	89.6%	90.2%
4S	88.2%	84.3%
2S-1	91.0%	89.4%
2S-2	88.0%	87.8%
3S-1	86.1%	85.3%
3S-2	88.2%	81.3%
3S-3	89.7%	86.4%
2S-1-2	76.1%	79.2%

5. CONCLUSIONS AND DEVELOPMENTS

The paper has proposed and evaluated a neural network architecture and supervised training algorithm, that can be used as a binary classifier. In particular, the system has been shown to provide an accurate means of classifying noisy and variable patterns, for both character recognition and truck-type identification problems. Compared to Hamming networks, it has the advantage of being able to group patterns into different classes along any set of boundaries. A variant of the system where the thresholds on the hidden neurons are all set to zero has been shown to provide superior performance if a comprehensive set of noise-free templates are available for network development.

Important attributes of the training algorithm are that it operates extremely rapidly (being orders of magnitude faster than other supervised schemes such as backpropagation), it circumvents the issue of how many hidden neurons to incorporate in a network, and it is guaranteed to converge on zero error (if required) for a set of unambiguous training patterns.

The problems considered above are nontransitory in nature, that is, the targeted solution does not vary with time. To accommodate problems where the solution changes, the training process needs to be applied continually in some way.

[a] Processing speed experiments were performed on a Gateway 2000 486/33C computing system programmed using Turbo Pascal.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press

The method currently being evaluated to handle this situation exploits the fact that the main part of learning a mapping function is done by the first hidden neurons in the network, with subsequent hidden neurons becoming less significant. First, a base network is trained as described in Section 2.3 until the error for a fixed set of training patterns is reduced to an acceptable level (possibly zero). The set of training patterns is then refreshed, either by updating the target vectors or by replacing whole training patterns (possibly selected at random) with ones that are more up-to-date. Once the training pattern set has been modified, additional hidden neurons are added until the errors that have just been introduced are back down to an acceptable level. The next occasion the training pattern set is modified, the additional hidden neurons are discarded, and a new set is added. If an excessive number of additional hidden neurons have to be added at any stage then the base network may need to be updated. This can be done by cutting back the end portion of the base network and discarding it along with the additional hidden neurons thereby allowing retraining of the network from an earlier stage. Training as such can continue in the background while the latest version of the network is still in use.

REFERENCES

Anderson, J.A., Silverstein, J., Ritz, S.A., & Jones, R.S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psych. Review 84*, 413–451.

Anderson, J.A. (1983). Cognitive and psychological computation with neural models. *IEEE Transactions on Systems, Man, and Cybernetics SMC-13*, 799–814.

Fahlman, S.E., & Lebiere, C. (1990). *The cascaded-correlation learning architecture*. Report CMU-CS-90-100. Carnegie Mellon University, Pittsburgh, PA.

Flood, I. (1991). A Gaussian-based neural network architecture and complementary training algorithm. *Proc. Int. Joint Conf. on Neural Networks*, I, pp. 171–176. IEEE and INNS, Singapore.

Gagarin, N., Flood, I., & Albrecht, P. (1994). Computing truck attributes with artificial neural networks. *Journal of Computing*, ASCE *8(2)*, 179–200.

Gagarin, N., Flood, I., & Albrecht, P. (1992). Weighing trucks in motion using Gaussian-based neural networks. *Proceedings of the Int. Joint Conf. on Neural Networks*, II, pp. 484–489, IEEE and INNS, Baltimore.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences (USA) 79*, 2554–2558.

Hopfield, J.J., Feinstein, D.I., & Palmer, R.G. (1983). Unlearning has a stabilizing effect in collective memories. *Nature 304*, 158–159.

Hopfield, J.J. (1984). Neurons with graded response have collective computational abilities. *Proceedings of the National Academy of Sciences (USA) 81*, 3088–3092.

Lippmann, R.P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine 4(2)*, 4–22.

Lippmann, R.P., Gold, B., & Malpass, M.I. (1987). A comparison of Hamming and Hopfield neural nets for pattern classification. Technical Report 769, MIT Lincoln Laboratory, Lexington, MA.

Poggio, T., & Edelman, S. (1990). A network that learns to recognize three-dimensional objects. *Nature 343*, 263–266.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing, Exploring the Microstructure of Cognition*, (Rumelhart, D.E. and McClelland, J.L., Eds.), Vol. 1, pp. 318–362. MIT Press, Cambridge, Massachusetts.

Tsoi, A.C. (1990). Comparison of the performance of three popular neural networks. *Proc. Second Int. Joint Conf. on Neural Networks*, pp. 707–711. IEEE, Washington DC.

Ian Flood is Director of the Fire Research Center and a faculty member of the M.E. Rinker, Sr. School of Building Construction, at the University of Florida, Gainesville. Prior to this, he held academic positions in the Department of Civil Engineering at the University of Maryland, College Park, and in the School of Building and Estate Management at the National University of Singapore. He has published extensively in the areas of construction management and computer-based modeling in engineering. His current areas of research include the development of artificially intelligent methods of modeling construction processes, neural network-based approaches to the weigh-in-motion of trucks, and the modeling of uncertainty in construction costs and schedules.

Nabil A. Kartam is an Associate Professor of Civil Engineering at Kuwait University. His current teaching and research are centered around the design, implementation, and validation of advanced intelligent information systems in civil engineering. Special research emphasis on supporting the design-construction integration throughout the project life cycle, contracts and claims management, value engineering, constructability improvement, and construction safety. Dr. Kartam has more than 15 years experience as an engineer, planner, professor, and consultant in U.S.A. and Kuwait. He achieved an M.S.E. degree in construction engineering and management in 1985 from the University of Michigan, Ann Arbor; an M.S. degree in computer science/AI and expert systems in 1988; and a Ph.D. degree in civil engineering in 1989 from Stanford University.

https://doi.org/10.1017/S0890060498123041 Published online by Cambridge University Press