A comparative study of neural network algorithms applied to optical character recognition

P. Patrick van der Smagt

Department of Computer Science and Mathematics
Universiteit van Amsterdam, Amsterdam, The Netherlands[†]
email: *smagt@fwi.uva.nl*

[†]Research carried out at the author's previous address:
Department of Computer Science and Mathematics, Vrije Universiteit, Amsterdam, The Netherlands, email: *smagt@cs.vu.nl*.

Abstract *– Three simple general purpose networks are tested for pattern classification on an optical character recognition problem. The feed-forward (multi-layer perceptron) network, the Hopfield network and a competitive learning network are compared. The input patterns are obtained by optically scanning images of printed digits and uppercase letters. The resulting data is used as input for the networks with two-state input nodes; for others, features are extracted by template matching and pixel counting. The classification capabilities of the networks are compared with a nearest neighbour algorithm applied to the same feature vectors. The feed-forward network reaches the same recognition rates as the nearest neighbour algorithm, even when only a small percentage of the possible connections is used. The Hopfield network performs less well, and overloading of the network remains a problem. Recognition rates with the competitive learning network, if input patterns are clustered well, are again as high as the nearest neighbour algorithm.*

1. Introduction

Since the rebirth of neural networks, pattern classification has proved itself a secured application (e.g. [1,2,3]; see also [4]). Such networks are often dedicated and incorporate some existing clustering or classification algorithm; in some cases, they form part of hybrid systems. Advantages of neural networks are adaptability, robustness, and ease of implementation (especially on parallel processing equipment).

In this paper three general purpose networks are investigated and compared on their classification capabilities. These networks are:

- **the feed-forward network** which incorporates a hyperplane separating technique. The coefficients describing the hyperplanes can be found using the *perceptron convergence procedure* [5] or a similar technique for linear devices, or the *back propagation rule* [6, 7, 8] for higher order devices;
- **the Hopfield network** [9] which incorporates an associative memory. Setting up the associations establishes an area of influence around each stored pattern, such that iteration from a test pattern which lies in this area of influence will result in the prototype pattern being returned;
- **competitive learning** which implements the idea of unsupervised pattern classification. A competitive learning network is used to find clusters in the input data. No external teacher is involved.

2. The optical character recognition problem

Pattern recognition systems consist of the following three subproblems [10]:

2.1 Image measurement

Images are obtained by optically scanning a page with disconnected printed characters. The characters, in point size 10 Times Roman font, are scanned with a resolution of 300×300 dots per inch and subsequently reduced by a factor four (2 times 2), resulting in an approximate size of 14×14 pixels per uppercase letter or 15×10 per digit (see figure 1).

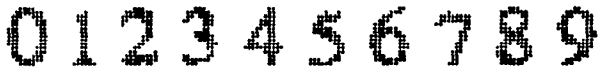

Figure 1. Ten of the scanned digits after the reduction phase.

2.2 Feature extraction

To filter out irrelevant data, feature extraction is performed using a set of novel features. Template matching is applied in the following form. Every image, fitted in a box in which the pixels on the edge are all "off," is divided in four equally sized quadrants. In each quadrant, twelve 2×2 masks are fitted on every possible position, and for each mask the number of perfect matches is counted. The area of the figure is also measured in each quadrant[1]. The resulting 13 features form 52 features for the entire image.

The templates used, shown in figure 2, are prototypes for horizontal, vertical, and diagonal line sections.

01	10	00	11	00	10	11	01	00	01	10	11
01	10	11	00	01	00	10	11	10	00	11	01
1	2	3	4	5	6	7	8	9	10	11	12

Figure 2. The templates used for feature extraction.

The templates or structuring elements can be interpreted as follows: the first two templates match vertical lines ('|'), the second pair horizontal lines ('—'), the next quadruple matches diagonal lines with positive gradient ('/'), and the last four match diagonal lines with negative gradient ('\'). This feature set, based on mathematical morphology principles [11], enables successful character recognition using a variety of classifiers. Although

[1]To prevent the areas of the image having too large an influence on the classification step, their values are halved.

these features do not allow scaling or rotation invariance, the system incorporates translation invariance.

2.3 Classification

For identification and classification of the feature vectors, optimum decision procedures must be determined. Each vector determined in step 2.2 must be assigned to a class or rejected. When complete knowledge about the patterns is available, decision functions can be constructed on basis of this information. Generally, however, only partial or no knowledge is available. For this case adaptive systems are needed. This paper describes the use of general purpose neural networks for adaptive pattern classification.

3. Description of the algorithms

3.1 Nearest Neighbour

The performances of the various neural network algorithms that are used are compared to the performance of the K-nearest-neighbour (K-NN) rule [10] under the same conditions. Since the prototype classes are usually small, typically containing several tens of characters, K is set to 1, such that each test pattern is assigned to the class of the nearest training element.

3.2 Feed-forward network

The general topology of what we call a K-layer[2] feed-forward network is depicted in figure 3.

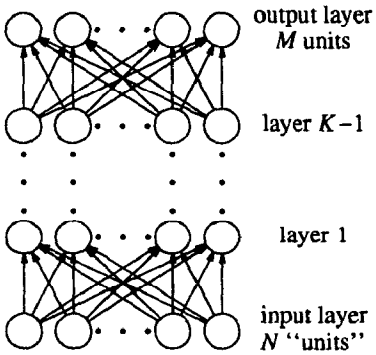

Figure 3. The general topology of a K-layer feed-forward network. There are N input units and M output units. Note that the input layer is not counted.

The weights and biases of each unit j in layer 1 describe a hyperplane in N-dimensional space. This hyperplane divides the input patterns in two classes, indicated by the activation value a_j. It is calculated as

$$a_j = f(\sum_{i=0}^{N-1} w_{ji}x_i + bias_j)$$

where x_i is the activation value of input i, w_{ji} the weight from input unit i to unit j, and $bias_j$ is a constant that is added to the total input of j. Finally, f is the activation function, bounding the activation value to the range [0, 1].

In a feed-forward network, a unit in layer 2 represents a convex region in hyperspace, enclosed by the hyperplanes determined by the units of layer 1. When three layers are used, any arbitrary shape in N-dimensional space can be enclosed.

The weights and biases of the units are determined as follows. At the input nodes, a pattern is clamped. The activities of the units in successive layers are computed, ending with the output units. Next, the output activities are compared with the *desired* outputs $\boldsymbol{d} = (d_1, \cdots, d_M)$. The difference $d_j - a_j$ is used to compute the amount by which the weights must be changed:

$$\Delta w_{ji} = \gamma \delta_j a_i \qquad (1)$$

where a_i is the activity of node i in the layer directly below the output layer, γ is a constant called the *gain* term, and

$$\delta_j = (d_j - a_j) f_j'(x_j + bias_j)$$

Next, δ is calculated for the hidden units directly below the output units by combining the δ's from the output units:

$$\delta_j = f_j'(x_j + bias_j) \sum_k \delta_k w_{kj}$$

and equation (1) is applied to these units as well. This process is repeated up to the input units. This learning rule is called the *generalized delta-rule* or *back propagation rule*. Biases, which can be implemented as a weight from a dummy unit that is always on, can be learned using the same rule. The above formulas result from interpreting the difference (or error) between the current output and desired output as a function of weights and biases, and performing minimization of this function [12].

Most feed-forward experiments described in this paper concern one- or two-layer feed-forward networks. Learning parameters $\gamma = 0.2$ and $\nu = 0.9$ are used; NETtalk [13] was trained using the same parameters. The above-mentioned learning rule is enhanced by using a momentum term as proposed in [14]:

$$\Delta w_{ji}(t+1) = \gamma \delta_j a_i + \nu \Delta w_{ji}(t)$$

which speeds up the gradient descent search considerably. The activation function used is a sigmoid function:

$$f(x) = \frac{1}{1+e^{-\lambda(x+bias)}}$$

with first derivative

$$f'(x) = \lambda f(x)(1-f(x))$$

For most experiments, $\lambda = 1$ is chosen. Furthermore, sparsely connected networks are investigated, both networks in which a random part of the connections is kept and those which are connected to fit the problem posed. Also, network immunity against hardware deterioration is tested.

3.3 Hopfield network

The Hopfield network consists of a set of fully connected two-state neurons (figure 4) [9].

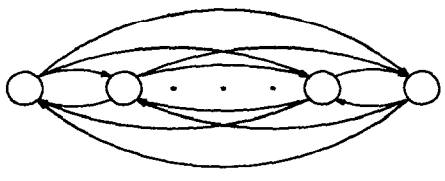

Figure 4. The general topology of a Hopfield network.

Since the neurons in the basic Hopfield network have binary values (viz. +1 and −1, which may be interpreted as the neuron being "on" or "off," respectively[3]), an alternative feature set is

[2] Since the neurons in the input layer do not compute their values but are simply clamped, the input layer is not counted. E.g., a network without hidden units is called a 1-layer network.

[3] In his original paper, Hopfield used the values 1 and 0, but using +1 and −1 for activation values presents some advantages [15].

used for training and testing, consisting of the concatenation of the raster-scan bits from the window of the binary representations of the characters. In particular, 15×10 bit binary images of digits (figure 1) and 14×14 bit images of uppercase letters are used.

In the basic model, all neurons are connected to each other. The connection weights between neuron i and j are arranged in a matrix $\boldsymbol{W} = (w_{ij})$. Each neuron randomly and asynchronously updates its activation value a_i according to the rule

$$a_i(t+1) = \text{sign}\left[\sum_j w_{ij} a_j(t)\right] \tag{2}$$

In other words, if the total input of a neuron is positive, the neuron should come on or stay on; if it is negative, the neuron should go off or remain off. The formula assumes a threshold of the neurons at 0, but any other value may be chosen.

Using this update rule, the network will always evolve to a stable state, i.e. reach a state in which for all units equation (2) holds true. This can be proved by showing that under the update rule, the *computational energy*

$$E = -\frac{1}{2}\sum_{i\neq j}\sum w_{ij} a_i a_j \tag{3}$$

is monotonically decreasing and bounded from below. A pattern is called stable if the network, when the pattern is clamped, is in a stable state.

The memory vectors can be stored with the Hebb rule [16]:

$$w_{ij} = \begin{cases} \sum_{p=0}^{M-1} x_i^p x_j^p & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

where M is the number of patterns to be stored, and x_i the i^{th} element of pattern $\boldsymbol{x}$. This rule thus adds one to the weight of a connection if two connected neurons have the same activation value, otherwise subtracts one.

It is observed that frequently, when multiple vectors are stored, many of these vectors fail to be fixed points. To improve upon the stability of the patterns, the rule proposed by Bruce *et al.* [17] is applied.

Another problem is that, besides the stored states, many spurious patterns are stable. *Unlearning* [18] is used to reduce the influence of spurious stable states by repeatedly applying the Hebb rule in reverse to stable states reached from random initial states, but with a very low (un-) learning factor.

3.4 Competitive learning

Competitive learning is an unsupervised neural network algorithm that can be used to find clusters in input patterns. The general model, as presented in [19], consists of an N-unit input layer and an M-unit output layer, with M equal to the number of required output classes. All input units are connected to all output units, with associated weights initialized to random values. Weights and input vectors must be kept normalized.

In the learning stage, when a pattern is clamped, all output units determine their net inputs by calculating the dot or inner products of the input vector and their weight vectors. The output unit with the highest net input "wins" the competition, meaning that its weight vector is the most similar to the input vector. This weight vector is then rotated in the direction of the input vector, by adding a fraction γ of the difference between input vector and weight vector to it [20]:

$$\boldsymbol{w}_j(t+1) = \frac{\boldsymbol{w}_j(t)+\gamma\,[\boldsymbol{x}(t) - \boldsymbol{w}_j(t)]}{\|\boldsymbol{w}_j(t)+\gamma\,[\boldsymbol{x}(t) - \boldsymbol{w}_j(t)]\|} \tag{4}$$

in which γ is called the *learning rate*. To prevent that some output units never win the competition, the weight vectors of the losing units are also adapted using equation (4), but now with a *leaky learning rate* κ instead of γ, where $\kappa \ll \gamma$.

Kohonen [21], who presents an unsupervised learning network as an explanation of the existence of ordered maps in the brain, orders the output units such that not only the weight vectors to a winning unit are affected, but those to its neighbours as well (figure 5).

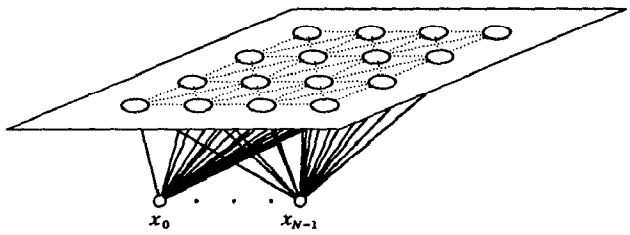

Figure 5. A typical Kohonen network. Here, the output units are ordered in a two-dimensional array. The neighbours of a winning unit are adapted with a learning factor inversely related to their distances to the winning unit.

4. Results

All recognition rates reported apply to recognition of sets different from the training sets. Since small test sets are used, typically containing only several tens of characters, the percentages reported must be regarded as being indicative only.

4.1 Nearest neighbour

Earlier work [22] shows that recognition rates using a nearest neighbour classifier are 98% for the uppercase letters and 100% for the digits. As a comparison, when using a feature vector composed from the concatenation of the raster-scan bits from the window of the binary image (such as used for the Hopfield network experiments), rates of 85% and 90% are achieved, respectively. Further confidence in the proposed feature set is expressed by the mean relative distance to the nearest bad choice, i.e.,

$$\text{error ratio} = \frac{\text{distance to nearest pattern class}}{\text{distance to second nearest pattern class}}$$

provided the nearest pattern class is the correct one. This error ratio is 0.56 for uppercase letters and 0.47 for digits, whereas it is 0.87 and 0.77 for the binary feature vector.

4.2 Feed-forward

When a feed-forward network is used for classifying patterns that differ from the test set, recognition rates can be as high as those obtained with nearest neighbour[4]. There are, however, some parameters that have to be considered.

4.2.1 Number of output units

When M pattern classes have to be distinguished, the two most obvious choices for output patterns are M-bit binary patterns or

[4]Here we consider an input pattern rejected when the difference between the highest and second highest output activation values does not exceed 0.2.

$\lceil \log M \rceil$-bit binary encoded patterns. Training a network with the latter proceeds faster because one training cycle takes less time, and less cycles are needed due to the smaller number of constraints that have to be satisfied. For the same reason, however, recognition of a different test set is considerably less good, generally not exceeding 70%. Therefore the experiments reported all have one output neuron assigned to one pattern class.

4.2.2 Number of hidden units

When the input patterns are linearly separable no hidden units are needed. Relative to the number of input patterns, the dimensionality is very high and it is very unlikely that the patterns are not linearly separable. Thus a 1-layer feed-forward network generally achieves the same recognition rates as nearest neighbour classifier. However, introducing hidden units presents the advantage of reduced susceptibility to deteriorating hardware. This is shown in section 4.2.4.

When a hidden layer is present, recognition rates depend on the number of hidden units present. As a rule of thumb, the greater the number of units, the better the recognition. This is due to the better distribution of "knowledge." Using too many hidden units, however, may increase the complexity of the error surface such that the training patterns cannot be separated – the system gets stuck in local minima; also, there is lower bound of $\lceil \log M \rceil$ hidden units when separating M pattern classes[5].

4.2.3 Training sparsely connected networks

Since such a vast number of connections exist, it is highly probable that not all connections are equally essential. Simulations show that recognition is not seriously affected when connections are cut before training.

When using no hidden units, a missing connection means missing information for some output unit. Due to the great number of inputs many connections still can be cut, but a network with hidden units is less sensitive, especially when the hidden and output layers are kept fully connected. Figure 6 depicts the decrease in recognition rates when connections are randomly cut for this latter case. When the connections between more than two successive layers are cut, a minimum of around 70% of the connections must be kept. Besides random cutting, it is also possible to assign specific hidden units to specific portions of the input. For example, a network with eight hidden units, which are pairwise fully connected to each of the four quadrants, performs just as well as a fully connected eight-hidden unit network: 98% recognition of the digits. Using one hidden unit per quadrant does not suffice; also, using five hidden units with one assigned to each input pattern "feature" (one for each direction and one for the area) gives no spectacular results.

4.2.4 Network immunity against degrading hardware

Neural networks are often presumed to have a certain immunity against "hardware faults." Starting from a fully connected network, connections or units are destroyed after the teaching phase has been successfully completed.

As in the previous section, a network with hidden units gives better results when the hidden and output layers are constrained to be fully connected. Figure 7 shows the results. As before, the more hidden units are used (i.e. the more connections are present), the greater the immunity of the network. A network without hidden units performs less well than a network in which

[5]For all practical purposes. Although activation values can have any value in the range [0,1], the form of the sigmoid function dictates that they be either 0.0 or 1.0.

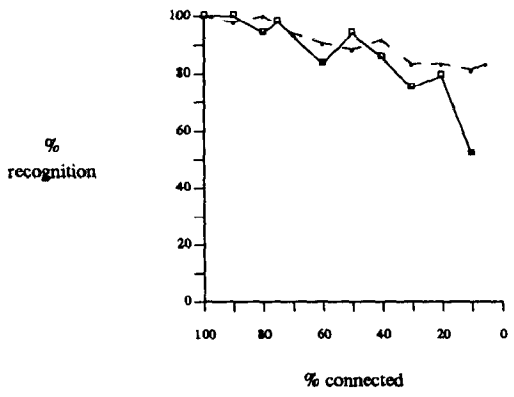

Figure 6. Recognition rate drop related to connectivity. The solid curve is for a network with ten hidden units, the dashed for a network with seven hidden units. Note that the hidden and output layers are kept 100% connected. Since there are much fewer output units than input units, there are much fewer connections between layer one and two than between the input layer and layer one.

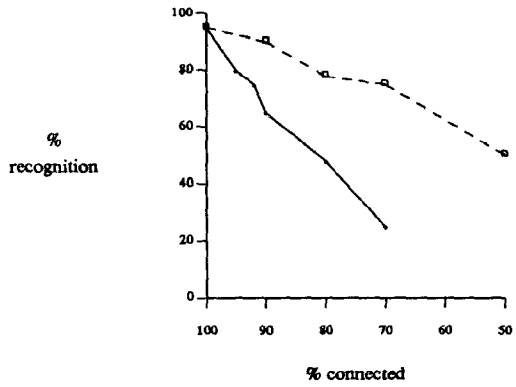

Figure 7. Recognition rates after damaging weights for a network with six hidden units. The dashed curve depicts the case in which only connections between input and hidden units are cut; for the solid curve, all connections were cuttable. Note that the horizontal scale is a 50–100 one and not 0–100.

the hidden and output layers remain connected, and graceful degradation is not attained.

The number of hidden units that are allowed to fail depends on the number of hidden units used. The optimality of a solution found by the network can be illustrated by running the network on binary valued input patterns. When only $\lceil \log M \rceil$ hidden units, which tend to have activation values of 0.0 or 1.0 with binary input patterns, are used, all units are necessary to distinguish between M input patters. Now suppose there are $\lceil \log M \rceil + \delta$ hidden units available. These hidden units are used optimally when the Hamming distances between their activation values for patterns p_i and p_j for each i, j such that $i \neq j$ are equal to each other and have a maximal value $2\delta+1$. In that case, δ hidden units, no matter which, may fail.

We observe that the networks often operate near this optimal case. As an example, we trained a network with ten hidden units on the binary images of the ten digits. The network found hidden unit activations with a Hamming distance 3 between every pair of hidden unit activations, which is not optimal but allows an arbitrary hidden unit to be removed[6].

[6]The Hamming distance between two binary words is defined as the number

4.2.5 Interpretation of the weights

Networks without hidden units

In a sense, the weights (and biases) found by the network reflect the input patterns. When no hidden units are used, each of the M outputs learns to react to the patterns that belong to the class associated with that output. That this is so can be seen by looking at the Hinton diagrams[7] for the weights. As an example, figure 8 depicts Hinton diagrams for a one-layer feed-forward network trained on binary patters similar to those of figure 1.

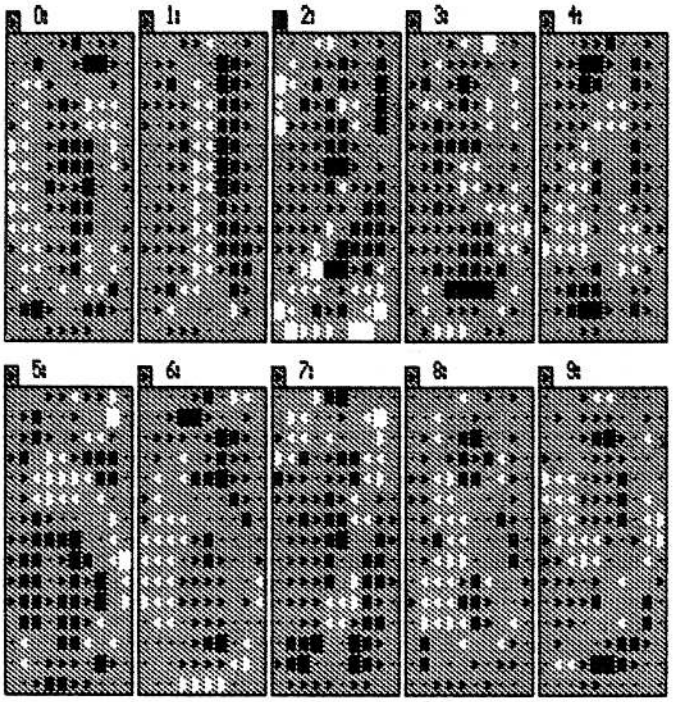

Figure 8. Hinton diagrams for a one-layer feed-forward network.

As can be seen from these Hinton diagrams, the network reacts on the *differences* between input patterns.

Networks with hidden units

By examining the Hinton diagrams for feed-forward networks trained on the 52 feature sets, it can be easily seen that the similarities between the structuring elements are recognized by the network. For example, the weights that react on matchings of templates 1 and 2 in figure 2 are usually nearly equal; both template 1 and 2 are vertical line detectors. Consequently, to show the Hinton diagrams for the networks trained on the 52 feature set, weights which code for the same feature are added together. The resulting diagrams are ordered as shown in figure 9.

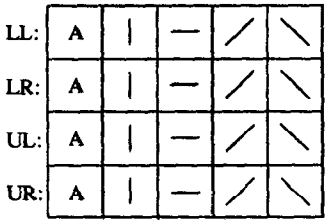

Figure 9. Meaning of the weights in Hinton diagrams for 52 feature set input networks. Every row represents a quadrant in the image, and every element in this row the interpretation of the input stimulus. 'A' stands for area, 'LL' for the lower left quadrant, and so on.

Figure 10 shows Hinton diagrams for a two-layer network with ten hidden units, trained on the uppercase letter patterns.

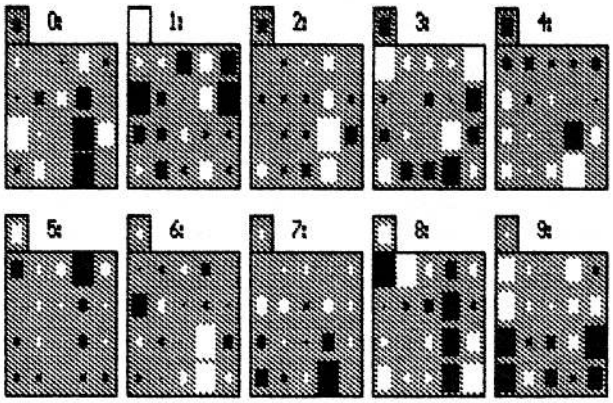

Figure 10. Hinton diagrams for a feed-forward network with one layer of ten hidden units, trained on uppercase letter patterns. The weights shown are those from the input to the hidden units.

It is worthwhile examining some specific diagrams. For example, hidden unit 1 has a large negative weight for '\' in the lower right quadrant. It has a positive bias, so its quiescent state is "on" and goes "off" when an A, B, K, Q, R, S, W, or X is clamped. Hidden unit 2 acts as a vertical line detector in all quadrants (since small templates are used, templates 1 and 2 react to 60° lines and templates 3 and 4 on 30° lines as well). Hidden unit 9 mainly reacts on area difference between the upper and lower half of the character. It goes "off" for a C, F, P, T, V, W, X, and Y, most of which have a darker upper half.

The higher order structure which is thus found by the hidden units can be used for further analysis of the input data.

4.3 *Hopfield networks*

When patterns are stored in the Hopfield network, it is imperative that all these patterns be stable to recall them. Tests have shown that using the Hebb rule for storing random patterns, about 15% of the storage capacity of the network can be used before recall error is severe [9]. Keeping in mind that the complement of a stable state is also stable, only half of this 15% can be used effectively.

The binary patterns, described above, on which the Hopfield

---

of bits in which they differ [23]. Provided the Hamming distance between every pair of codewords is at least $2d+1$, up to d errors in a codeword can be corrected. In our case, a Hamming distance of 3 allows one hidden unit to flip its activation value without problems occurring. An optimal separation distance which can be found for at least ten codewords in ten bits is 5:

1111100000	1001001100
1100011010	0100100110
0110001101	1010000011
0011010110	0101010001
0001101011	0010111000
1000110101	1111111111

We kindly thank dr. Evert Wattel, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam for finding this code.

[7] Hinton diagrams are set up as follows. Each large shadowed rectangle represents the weights to a specific hidden or output unit; each small square in this rectangle is a weight from a unit in the previous layer to this unit. A white square denotes a positive weight, a black square a negative one. The size of the square indicates the size of the weight. The bias is depicted in a small shadowed rectangle above the large one.

network is tested here, are much more correlated than random patterns. They are not evenly distributed in the input space. In fact, instead of 15% only 10% of the storage capacity of the network can be used. To solve this problem, the algorithmic enhancements described below are used [17]:

Algorithm 1:

(1) Given a starting weight matrix w_{ij}, define a correction ε_i for each pattern to be stored such that

$$\varepsilon_i = \begin{cases} 0 & \text{if } a_i \text{ is stable} \\ 1 & \text{if } a_i \text{ is not stable.} \end{cases}$$

(2) Now modify w_{ij} by $\Delta w_{ij} = a_i a_j(\varepsilon_i+\varepsilon_j)$ if $i \neq j$.

(3) Repeat this whole procedure until the patterns are stable.

It is claimed that the algorithm will find a solution in finitely many steps, provided it exists.

4.3.1 Accessibility of the stored patterns

When all the patterns are stored and stable, it is desirable to let those patterns (and their inverses) to be the only stable states of the system. Every stored pattern can be seen as a "dip" in energy space (equation (3)), and is surrounded by a basin of influence. In an ideal Hopfield network, all stored states have the same energy, and their basins of influence fill up the whole energy space. The *accessibility* [18] of the stored patterns expresses the fraction of times a nominally assigned stable state is reached from a random state.

Simulations show that when the Hebb rule plus the stabilization procedure for storing ten digits is used, an accessibility of around 60% is reached; for the uppercase letters, this is 50%. To improve upon the accessibility of the patterns, unlearning is applied. Thus the very low energy minima are "lifted," resulting in a more even distribution [24]. Also, the distribution among the states that are reached is better.

When unlearning is applied too often, assigned stable states are destroyed and the performance of the network deteriorates. In this particular case, best results are obtained by unlearning some 100 times with a 0.05 factor.

4.3.2 Adapted learning rule

In the configuration of the network described so far, all neurons are essentially indistinguishable from each other in that the spatial information present in the input patterns is not used. However, that information can be incorporated in the learning rule. The proposed learning rule sets the weights proportional to the distance $\delta(i,j)$ between neurons i and j:

$$w_{ij} = \begin{cases} \delta(i,j) \sum_{p=0}^{M-1} x_i^p x_j^p & \text{if } i \neq j \\ 0 & \text{if } i=j \end{cases} \qquad (5)$$

For reasons of simplicity, the Manhattan distance function $\delta(i,j) = |i-j|$ is used.

Since the input patterns used here consist of large clusters of "on" or "off" pixels, the proposed rule increases the influence of such clusters on the stability of all patterns. When such clusters are shared by all or most of the patterns, the synaptic strengths from and to these clusters are very large, increasing the stability of all stored patterns. In this case, the network in which the digit patterns are stored with rule (5) reaches an accessibility of 90% instead of 60%, and recognition rates rise accordingly. However, overloading the network has the opposite effect, and the original Hebb rule is preferred.

4.3.3 Recognition rates

In the original configuration, recognition rates are 40% for the uppercase letters and 75% for the digits. Unlearning improves this to 80% and 90%, respectively, which is not as good as nearest neighbour classification on the same input vectors. When the adapted Hebb rule is used, 55% of the uppercase letters and 95% of the digits is correctly classified.

4.4 Competitive learning

To monitor the progress in teaching competitive learning network, an expression for the error must be found. Most straightforward is the sum squared error [12]

$$Error = (w - x)^2$$

The competitive learning algorithm is employed as follows. Initially, both the learning rate γ and leaky learning rate κ are set to 0.5. This has the effect of rotating the weight vectors in the direction of the pattern vectors (see figure 11).

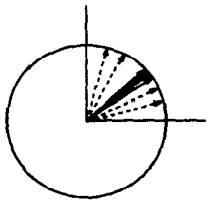

Figure 11. Two-dimensional representation of input pattern vectors (solid lines) and weight vectors (dotted lines). All weights and inputs are positive. As mimicked in the figure, the input patterns all lie comparatively close to each other, whereas the weight vectors are evenly distributed. When only the winning unit would learn with this type of input, all the other units would remain inactive and never change. Leaky learning is needed.

When no further progress is made, γ is increased by a small amount while κ is decreased by the same amount. This procedure is repeated until γ=1.0 and κ=0.0. Figure 12 depicts how clustering proceeds. It must be stressed that the input data consists of clusters of only a few patterns. When larger clusters have to be formed, a finer tuning of the learning parameters is necessary.

Perfect separation is always obtained with one set of digits or uppercase letters. However, the Kohonen neighbourhood training method sometimes maps two different input patterns on one output unit. Since the network is, in fact, a clustering device, multiple sets of input patterns can be taught. Often clustering works well, especially with the digit input patterns.

Since the patterns are literally stored in the weights, recognition rates are precisely those obtained with nearest neighbour classification, provided that the input patterns are perfectly separated.

4.4.1 Reduced connectivity

Training a competitive learning network with fewer connections gives results similar to those obtained with the feed-forward experiments. However, there is a greater susceptibility to failing connections. Figure 13 depicts recognition rates.

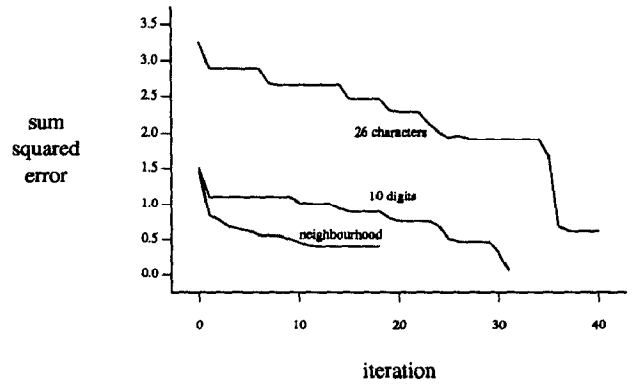

Figure 12. Reduction of the sum squared error when training a competitive learning net with 26 characters (upper line), 10 digits (middle line) and 10 digits with neighbourhood training (lower line). In the former two experiments, when the curve remains flat, γ and κ are adapted resulting in a sudden drop in the error curve as can be clearly seen. In the latter, such a mechanism is not needed.

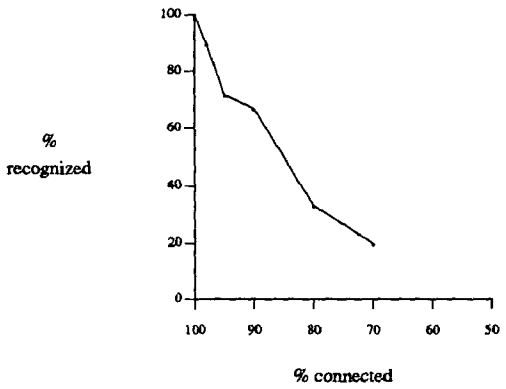

Figure 13. Recognition rates after damaging weights of a competitive learning network.

5. Conclusions

Optical character recognition (OCR) appears to be a good application for neural network classification. The advantages of using neural networks for OCR include

(a) automatic training and retraining;

(b) graceful degradation and robust performance;

(c) potential for parallelization; and

(d) potentially less storage.

We consider points (a) and (b) to be the major fortes of neural networks for this as well as other applications. The advantages of alternative approaches, in particular K-nearest-neighbour, include

(a) improved performance;

(b) simple implementation and training; and

(c) known design methodology.

The feed-forward network has the additional advantage that it is relatively immune against failing units and connections. The best recognition rates are realized with a one-layer feed-forward network with one output unit reserved for each input class. Similar rates are obtained with a (much cheaper) network having less than 10 hidden units. In this configuration, the hidden units gather higher-level information about the input patterns, which could be used in a more advanced system.

Before the Hopfield network can be used as a pattern recognizer, problems of instability of stored patterns and stability of a large number of spurious patterns must be overcome. The Hopfield in its basic configuration network is much better suited for storage of random patterns than of patterns which are all much alike, such as optical images of printed characters. The network, being used as an associative memory, is capable of reaching acceptable recognition results when it is not overloaded and unlearning is applied.

The competitive learning algorithm could well be used to cluster large amounts of input data. Our tests were simple in the sense that each cluster typically contained just a few patterns. Further refinements of the basic competitive learning scheme will probably show it to be a viable clustering method in its own right or possibly a useful part of a larger neural network.

Acknowledgements

The research work on which this paper is based formed part of the master's thesis by Jim E. Stada and the author. I am therefore greatly indebted to Jim for his cooperation, co-writing earlier papers, and for extensively researching the Hopfield and competitive learning networks. Also, I would like to thank Robert A. Hummel for acting as our thesis supervisor, for many stimulating discussions, and for proofreading this and other documents. Finally, I am indebted to Floor van der Ham for literature references and hints and helps.

References

[1] D. MEHR AND S. RICHFIELD, "Neural net application to optical character recognition," in *IEEE First International Conference on Neural Networks*, ed. M. Caudill, 21-24 June 1987.

[2] K. FUKUSHIMA, "Neocognitron: a hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, pp. 119-130, 1988.

[3] M. FISCHLER, R. L. MATTSON, O. FIRSCHEIN, AND L. D. HEALY, "An approach to general pattern recognition," *IRE Transactions on Information Theory*, vol. IT-8, no. 5, 3-7 September 1962.

[4] W. H. HIGHLEYMAN, "Linear decision functions, with application to pattern recognition," *Proceedings of the IRE*, pp. 1501-1514, 1962.

[5] F. ROSENBLATT, *Principles of neurodynamics*, Spartan Books, New York, 1959.

[6] Y. LE CUN, "Une procedure d'apprentissage pour reseau a seuil assymetrique," *Proceedings of Cognitiva*, vol. 85, pp. 599-604, 1985.

[7] D. B. PARKER, "Learning-logic," TR-47, Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, Cambridge, MA, 1985.

[8] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS, "Learning representations by back-propagating errors," *Nature* , vol. 323, pp. 533-536, 1986.

[9] J. J. HOPFIELD, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, pp. 2554-2558, 1982.

[10] J. T. TOU AND R. C. GONZALEZ, *Pattern recognition*

principles, Addison-Wesley Publishing Company, Inc., 1974.

[11] J. SERRA, *Image analysis and mathematical morphology,* Academic Press, Inc., 1982.

[12] G. E. HINTON, "Connectionist learning procedures," CMU-CS-87-115 (version 2), Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA, 1987.

[13] T. J. SEJNOWSKI AND C. R. ROSENBERG, "NETtalk: a parallel network that learns to read aloud," JHU/EECS-86/01, The John Hopkins University Electrical Engineering and Computer Science Department, 1986.

[14] J. L. MCCLELLAND AND D. E. RUMELHART, *Explorations in parallel distributed processing: Computational models of cognition and perception,* The MIT Press, 1988.

[15] P. P. VAN DER SMAGT AND J. E. STADA, *A view on neural networks,* 1989. Manuscript

[16] D. O. HEBB, *The organization of behaviour,* Wiley, New York, 1949.

[17] A. D. BRUCE, A. CANNING, B. FORREST, E. GARDNER, AND D. J. WALLACE, "Learning and memory properties in fully connected networks," in *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* ed. J. S. Denker, 1986.

[18] J. J. HOPFIELD, D. I. FEINSTEIN, AND R. G. PALMER, "'Unlearning' has a stabilizing effect in collective memories," *Nature,* vol. 304, pp. 159-159, 1983.

[19] D. E. RUMELHART AND D. ZIPSER, "Feature discovery by competitive learning," *Cognitive Science,* vol. 9, pp. 75-112, 1985.

[20] T. KOHONEN, *Self-organization and associative memory,* Springer-Verlag, Berlin, 1984.

[21] T. KOHONEN, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics,* vol. 43, pp. 59-69, 1982.

[22] P. P. VAN DER SMAGT AND J. E. STADA, *Aspects of printed character recognition,* 1989. Manuscript

[23] T. M. THOMPSON, "From error-correcting codes through sphere packings to simple groups," in *Number twenty-one from The Carus Mathematical Monographs,* The Mathematical Association of America, 1983.

[24] R. J. SASIELA, "Forgetting as a way to improve neural-net behaviour," in *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* ed. J. S. Denker, 1986.