TRANSFORMATIONS AND DISTORTIONS TOLERANT RECOGNITION OF NUMERALS USING NEURAL NETWORKS

R. P. SRIVASTAVA

Towson State University, Towson, Maryland

ABSTRACT

This paper presents a multilayered artificial neural network model to recognize numerals independent of their sizes, positions, and orientations. We used a modified backpropagation algorithm to train the network. We preprocessed the numerals for "feature" extraction by calculating their moments. The moments are invariant under translation, scaling, and rotation transformations. We used the moments as input to the network rather than the digitized pattern itself. The network was able to recognize transformed and slightly deformed numerals. Parallel computational capability of the network makes it an attractive alternative for real-time commercial applications.

1. INTRODUCTION

The recognition of numerals has several practical applications including optical page readers, and mail sorters. In order to provide a practical system the recognition algorithm should be tolerant of small distortions and possible transformations. Developing such algorithm is a difficult problem in designing pattern-recognition systems. Many authors [1,2,3,5,9,16] have tried to solve this problem but no satisfactory general theory exists. Often, the methods are based on extracting "features" which are invariant under transformations. The main difficulty in using the conventional techniques is that they are not fault tolerant. Neural networks have been found to be fault tolerant in pattern recognition[3,4,5,6,7,8,9,12]. Fukushima and Miyaki [3] used a multilayered neural network, called "neocognitron," to recognize numerals. Their method is tolerant of small distortions and shifts in position but it is quite difficult to train a "neocognitron" to include rotational invariance as well. Burr[4] used neural networks to design a text reader. The pre-processing done by Burr to produce a compact input pattern requires a substantial computation time. Rajavelu, Musavi and Shirvaikar[5] used Walsh functions to extract pattern features which were inputs to a neural network. In this paper we have investigated the possibility of using invariant moment functions as input to a neural network in recognizing hand written numerals. Kulkarni, Yap, and Bayers [6] used moments as input to neural networks for analyzing aircraft images. We used backpropagation [11,13] algorithm to train a multilayered neural network.

2. SYSTEM CONFIGURATION

Numbers 0 to 9 were digitized on a grid of 50 * 60. A contour polygon was made for each number and then it was filled using a polygon filling algorithm. A pixel was weighted as one if it was inside the polygon, otherwise it was weighted as zero.

We preprocessed the image for feature extraction. Preprocessing has several advantages including:

1) It reduces the network size and requires less memory space to process information.

2) It reduces the training time and increases the chances for convergence.

3) It extracts features which are invariant under transformations. This is important for building a practical pattern recognition system.

The preprocessor calculates a set of seven moments which are invariant under transformations. These moments represent "features" of the image. We used a multilayered network with one hidden layer (fig. 1). There are seven neurons in the input layer, one for each moment. The network output was matched against a target vector representing the numeral being considered. We experimented with two target vectors for each numeral, one using unary and the other binary representation (fig. 2). The output layer consists of ten and four neurons in unary and binary targets respectively.

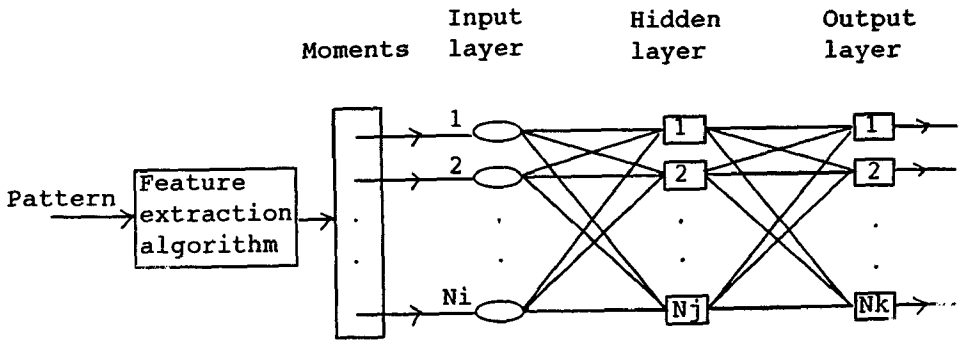

figure 1

The selection of the number of neurons in the hidden layer was based on experimenting with several cases starting with the sum of the input and output layers, and later reducing it until optimal. There are 14 and 10 neurons in the hidden layer of unary and binary targets respectively. The binary representation took less training time but it was not as accurate in recognizing an unknown pattern as unary representation.

	representation	
digit	unary	binary
0	0000000000	0000
1	0000000001	0001
2	0000000010	0010
3	0000000100	0011
4	0000001000	0100
5	0000010000	0101
6	0000100000	0110
7	0001000000	0111
8	0010000000	1000
9	0100000000	1001

Figure 2

3. BACKGROUND INFORMATION

3.1 Backpropagation Model:

The back propagation model employs an iterative gradient descent algorithm designed to follow the slope of the error surface, constantly adjusting the weights to minimize the mean square error between the actual output of a multilayer network and the desired output. To explain this model we are following the notations and methodology given in Wasserman [10].

Let us consider the neuron j shown in figure 3. If the vector $O = (O_1, O_2, O_3 ... O_n)$ represents an input to neuron j, and W_{ij} represents a weight for the signal going to neuron j from input i, then the output is given by

$$OUT_j = F(NET_j) = 1/(1 + e^{-NET_j}) \text{ where}$$

$$NET_j = \sum_{i=1}^{n} O_i W_{ij} \qquad \ldots (1)$$

The function $F(x) = 1/(1 + e^{-x})$ is a sigmoidal logistic function.

A network of three layers is shown in figure 1. We apply a set of known inputs and adjust the weights of the hidden layer and the output layer such that error is minimum. The error is calculated by comparing the actual output with the desired output. Once the network is trained with known input-output pairs, it can recognize unknown input patterns. At the start of the training process, all the weights are initialized with random values between -1 to +1.

Let us consider figure 4 to explain the calculations involved in backpropagation algorithm. Let i,j,and k be the input, hidden and output layers respectively, and let N_i, N_j, and N_k be the number of elements in layers i, j, and k respectively. Let r be an element in layer i, p be an element in layer j and q be an element in layer k. The other notations are:

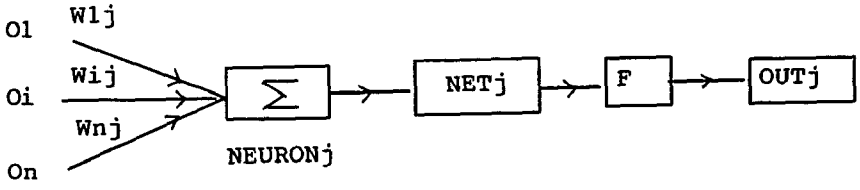

figure 3

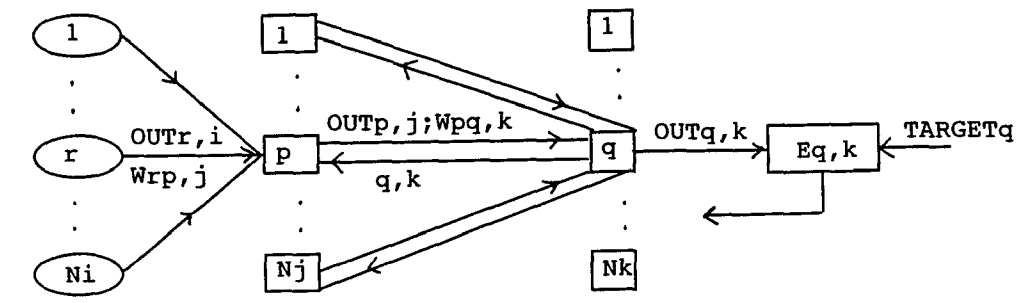

Figure 4

$OUT_{r,i}$ = output of element r in layer i,
$W_{rp,j}$ = Weight of layer j for the signal going from element r to element p,
$E_{q,k}$ = Error in element q in layer k,
$TARGET_q$ = Target value of element q,
$\delta_{q,k}$ = Error function for element q in layer k.

We follow the following steps to train the network.

1. Apply an input vector from the training pair to the network input layer. Output of input layer is the same as its input.
2. Calculate the output of hidden layer for each neuron using equation (1). Output of input layer is input to hidden layer.
3. Calculate the output of the output layer for each neuron using equation (1). Output of hidden layer is input to output layer.
4. Compare the calculated output with the target output for each neuron in output layer, and calculate the error.
5. If error is less than the tolerance for each neuron then go to step 11 and exit. The tolerance may be set to 0.1.
6. Calculate the error function $\delta_{q,k}$ for each neuron q in output layer k using the following equation (3)

$$E_{q,k} = (TARGET_q - OUT_{q,k}) \quad (2)$$

$$\delta_{q,k} = OUT_{q,k}\,(1 - OUT_{q,k})\,E_{q,k} \quad (3)$$

7. Calculate the error function of $\delta_{p,j}$ for each element p in the hidden layer j using equation (4).

$$\delta_{p,j} = OUT_{p,j}\,(1 - OUT_{p,j}) \left(\sum_{q=1}^{N_k} \delta_{q,k}\, W_{pq,k} \right) \quad (4)$$

8. Adjust the weights for output layer k using equation (6) in one of the following cases.

CASE 1: Basic Algorithm.

$$\Delta W_{pq,k} = \eta\, \delta_{q,k}\, OUT_{p,j} \quad (5)$$

$$W_{pq,k}(N+1) = W_{pq,k}(N) + \Delta W_{pq,k} \quad (6)$$

where η is learning rate parameter which can be set to 0.5 at the start and later on adjusted.

CASE 2: With Momentum Term Added.

$$\Delta W_{pq,k}(N+1) = \eta\,(\delta_{q,k}\, OUT_{p,j}) + \alpha[\,\Delta W_{pq,k}(N)] \quad (5)$$

$$W_{pq,k}(N+1) = W_{pq,k}(N) + \Delta W_{pq,k}(N+1) \quad (6)$$

where α = momentum parameter.

CASE 3: With Exponential Smoothing Term Added.

$$\Delta W_{pq,q}(N+1) = \beta\, \Delta W_{pq,k}(N) + (1 - \beta)\delta_{q,k}\, OUT_{p,j} \quad (5)$$

$$W_{pq,k}(N+1) = W_{pq,k}(N) + \eta\, \Delta W_{pq,k}(N+1) \quad (6)$$

where β = exponential smoothing parameter.

9. Adjust the weight for the hidden layer j by using equation(8).

$$W_{rp,j} = \eta \delta_{p,j}\, OUT_{r,i} \quad (7)$$

$$W_{rp,j}(N+1) = W_{rp,j}(N) + \Delta W_{rp,j} \quad (8)$$

10. Go to step 2.
11. Stop iteration and exit.

3.2 Feature Extraction:

The recognition of geometrical patterns independent of position, size, and orientation can be accomplished using moment invariants. These moments uniquely determine a piecewise continuous function f(x,y) which has non-zero values only in a finite part of the x-y plane. If f(x,y) be a digital image in two dimensional space, then the moments of order (p+q) can be defined by [15]

$$M_{pq} = \sum_x \sum_y x^p\, y^q\, f(x,y) \quad for\ p,q = 0,1,2\ldots \quad (9)$$

The central moments can be expressed as

$$\mu_{pq} = \sum_x \sum_y (x - x_c)^p (y - y_c)^q f(x,y) \quad (10)$$

where $x_c = M_{10}/M_{00}$, $y_c = M_{01}/M_{00}$ (11)

The normalized central moments N_{pq} can be defined as

$$N_{pq} = \mu_{pq} / (\mu_{00})^\gamma \quad (12)$$

where $\gamma = (p+q)/2 + 1$ for $p+q = 2,3,...$ (13)

Hu [16] has shown that a set of seven moments can be derived which are invariant to translation, rotation, and scaling transformations. These seven moments are:

$\phi 1 = N_{20} + N_{02}$

$\phi 2 = (N_{20} - N_{02})^2 + 4N_{11}^2$

$\phi 3 = (N_{30} - 3N_{12})^2 + (3N_{21} - N_{03})^2$

$\phi 4 = (N_{30} + N_{12})^2 + (N_{21} + N_{03})^2$

$\phi 5 = (N_{30}-3N_{12})(N_{30}+N_{12})[(N_{30}+N_{12})^2 - 3(N_{21}+N_{03})^2]$
$+ (3N_{21}-N_{03})(N_{21}+N_{03})[3(N_{30}+N_{12})^2-(N_{21} + N_{03})^2]$

$\phi 6 = (N_{20} - N_{02})[(N_{30} + N_{12})^2 - (N_{21} + N_{03})^2]$
$+ 4N_{11}(N_{30} + N_{12})(N_{21} + N_{03})$

$\phi 7 = (3N_{21}-N_{30})(N_{30}+N_{12})[(N_{30}+N_{12})^2 -3(N_{21}+N_{03})^2]$
$+ (3N_{12}-N_{30})(N_{21}+N_{03})[3(N_{30}+N_{12})^2-(N_{21} + N_{03})^2]$

A statistical program based on these moments can be developed and it can recognize numerals but the problem is that if noise is introduced into the image, the results are not reliable. It is known that neural networks are not sensitive to noise so if moment invariants are used as preprocessed inputs to neural networks, results are expected to be more reliable.

4. NETWORK TRAINING AND TESTING

Invariant moments for the digits were calculated using the formula given in section 3.2. The moments have very small values so we took their logarithmic values (Table 1). These moments were used as inputs to a multilayered neural network. The network was trained with the original patterns of numerals. To test the accuracy of the network in the recognition phase we calculated the moments of patterns under scaling, translational and rotational transformations using normalized coordinates. The moments of the original and transformed images for digits 2 and 3 are given in Table 2 for example. It shows that the moments are invariant under transformations. Figure 5 shows the numbers 2 to 9 under different transformations.

We programmed three cases of backpropagation algorithm:

1: Basic algorithm [11]
2: Momentum factor added to the basic algorithm in calculation of the change in weights. [11]
3: Exponential smoothing factor added to the basic algorithm in calculation of change in weights [12].

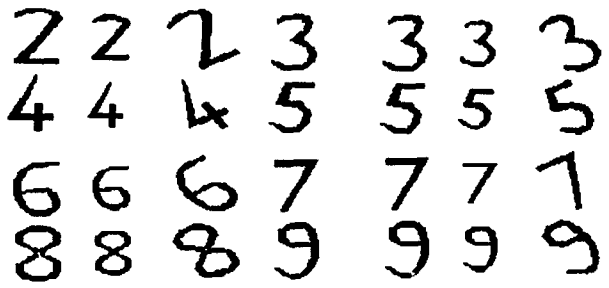

Figure 5

Digits	ϕ 1	ϕ 2	ϕ 3	ϕ 4	ϕ 5	ϕ 6	ϕ 7
0	0.2896724	1.8912424	3.0816103	3.7079442	7.5324230	5.0186122	7.1350254
1	0.1444406	0.3163917	2.4561373	2.6496310	5.2045640	2.8246001	6.2161687
2	0.1888211	1.0409897	2.0096739	2.5034514	5.8645744	3.1086389	4.7613598
3	0.1359538	0.6528863	1.3220851	1.9685698	3.7222681	2.2983096	3.8167549
4	0.3861797	1.6341399	1.3707669	2.0140326	3.7101914	2.9756539	4.5891420
5	0.2674511	1.0449647	3.3698149	2.9255948	6.1242224	3.4484834	6.4131852
6	0.2606826	1.3939269	1.6738422	2.0245292	4.0946032	5.1211048	3.9711671
7	0.1560557	0.6918714	0.6640617	1.2505373	2.3567873	1.6665286	2.3599277
8	0.3214875	1.4899748	4.3306973	4.3140626	9.3234240	5.1549184	8.6458210
9	0.2938891	1.2640356	1.7301497	1.9389643	3.9419327	2.9566812	3.9075031

Table 1

Digits 2:

Pattern	ϕ 1	ϕ 2	ϕ 3	ϕ 4	ϕ 5	ϕ 6	ϕ 7
original	0.1782108	0.7956657	2.5284379	2.6516904	5.2936189	4.9121131	5.5781130
trans. 0.2	0.1797515	0.7981299	2.5391998	2.6334396	5.2583461	4.1897512	5.6139249
scale 0.8	0.2000648	0.8591732	2.3372206	2.9122016	7.0309660	3.4452000	5.5371360
rotate π/6	0.1739678	0.7466299	2.2755938	2.5223495	4.9918835	4.3518786	5.1997384

Digits 3:

Pattern	ϕ 1	ϕ 2	ϕ 3	ϕ 4	ϕ 5	ϕ 6	ϕ 7
original	0.1359538	0.6528863	1.3220851	1.9685698	3.7222681	2.2983096	3.8167549
trans. 0.2	0.1359538	0.6528863	1.3220851	1.9685698	3.7222681	2.2983096	3.8167549
scale 0.8	0.1598016	0.6907020	1.4942709	2.1207083	3.9784019	2.4663624	4.2708248
rotete π/6	0.1032646	0.5299899	1.3127608	1.9181933	3.6848698	2.4416071	3.6835031

Table 2

For each of the above cases we experimented with two methods to encode a target vector: 1) Unary representation, and 2) Binary representation. In order to observe how the iterations are converging, we calculated root-mean-square error E:

$$E = \{[(T_1 - OUT_1)^2 + (T_2 - OUT_2)^2 + \ldots (T_n - OUT_n)^2]/n\}^{1/2}$$

where

$T = (T_1\ T_2 \ldots T_n)$ = Target Vector

$OUT = (OUT_1, OUT_2 \ldots OUT_n)$ = Output Vector

Each time we ran our training session we calculated the error (E) at the starting iteration of each round and plotted E vs N, where N = round number. In each round the system converged for all the digits under consideration one by one. We ran our experiments on an Intel 80386 microprocessor based microcomputer running at 20 MHz. We wrote the network software in Turbo Pascal. We ran our experiments first with only three digits 0, 1, and 2, and after studying the network behavior extended it to 10 digits. Some of our observations with three digits are given below, for example:

EXAMPLE 1: BINARY REPRESENTATION OF TARGET

Momentum parameter (α) = 0.9

Exponential smoothing parameter (β) = 0.5

Learning rate (η) = 0.2

Error tolerance limit (ϵ) = 0.2

Number of input layer neurons (N_i) = 7

Number of hidden layer neurons (N_j) = 10

Number of output layer neurons (N_k) = 4

Convergence data is shown in Table 3.

Algorithm	number of iterations	number of rounds	Time in minutes	Error curve
Basic algorithm	2311	169	2.65	
With momentum term	1429	144	1.83	
With exp. smoothing term	3883	191	5.51	

Table 3

Algorithm	number of iterations	number of rounds	Time in minutes
Basic algorithm	2059	157	4.36
With momentum	1604	168	4.13
With exp. smoothing	3171	165	9.8

Table 4

EXAMPLE 2: UNARY REPRESENTATION OF TARGET
Momentum parameter (α) = 0.9
Exponential smoothing parameter (β) = 0.5
Learning rate (η) = 0.2
Error tolerance limit (ϵ) = 0.2
Number of input layer neurons (N_i) = 7
Number of hidden layer neurons (N_j) = 14
Number of output layer neurons (N_k) = 10
Convergence data is shown in Table 4.

The basic backpropagation algorithm converged but it was taking a longer time and error curve was oscillating. By adding momentum term into the basic algorithm for weights adjustment we got faster convergence and smoother error curve. By adding the exponential smoothing term into the basic algorithm we got a smoother error curve but there was no saving in training time. On the basis of these observations we concluded that for our case the basic algorithm with momentum term was the best choice. We also experimented with different number of neurons (N_j) in the hidden layer. We found that the best results were obtained with N_j = 10 for the case of binary representation and N_j = 14 for the case of unary representation. We included all digits 0 to 9 and trained our network with both representations. It takes a shorter time to train with binary representation than with unary representation. After the training when we tested the accuracy of our network in recognizing transformed and slightly deformed patterns, we found that unary representation was accurate 96% times and binary representation was accurate 88% times. Here is our final data to train the network for all digits 0 to 9.
FINAL CONVERGENCE DATA:
Target representation = Unary
Momentum parameter (α) = 0.9
Learning rate (η) = 0.1
Error tolerance limit (ϵ) = 0.1
Number of input layer neurons (N_i) = 7
Number of hidden layer neurons (N_j) = 14
Number of output layer neurons (N_k) = 10
Total number of iterations = 19465
Total number of rounds = 572
Time taken to converge = 51 minutes.

5. CONCLUSION

We have demonstrated that a neural network using backpropagation training algorithm and a set of seven invariant moments can recognize numerals independent of their size, position, and orientation. The results show that inclusion of momentum term for weights adjustment improves the convergence time and makes the error curve smoother. A unary representation of the target takes a longer time to converge and needs more neurons in the output layer but its accuracy in recognizing untrained patterns is better than that of a binary representation. Slight variations in the input pattern do not affect the recognition accuracy of the network.

6. REFERENCES

1) Th. Pavlidis, "Algorithms for shape analysis of contours and waveforms", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol 2, pp 301-312, July 1980.
2) S.R. Ramesh, "A generalized character recognition algorithm: A graphical approach", Pattern Recognition, vol 22, No 4, pp 347-350, 1989.
3) K. Fukushima and S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position", Pattern Recognition vol 15, No 6, pp 455-469, 1982.
4) David J. Burr, "Experiments on neural net recognition of spoken and written text", IEEE Trans. on Acoustic, Speech, and Signal Processing, vol 36, No 7, pp 1162-1168, July 1988.
5) A. Rajavelu, M.T. Musavi, and M.V. Shirvaikar, "A neural network approach to character recognition", Neural Networks, vol 2, pp 387-393, 1989.
6) A.D. Kulkarni, A.C. Yap, and P. Byars, "Neural networks for invariant object recognition", Proc. IEEE, SAC'90 pp28-32, 1990.

7) Philip D. Wassarman, "Experiments in translating chinese characters using backpropagation", Proc. IEEE CompCon 88, pp 399-402, 1988.
8) Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, "Backpropagation applied to handwritten zipcode recognition", Neural Computation, vol 1, pp 541-551, 1989.
9) A.C.C. Coolen, and F.W. Kuijk, "A learning mechanism for invariant pattern recognition in neural networks", Neural Networks, vol 2, pp 495-506, 1989.
10) Philip D. Wasserman, "Neural computing - Theory and practice", pp 43-59, Van Nostrand Reinhold, New York 1989.
11) D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation", Parallel distributed processing, vol 1, pp 318-362, MIT Press, Cambridge, MA, 1986.
12) T.J. Sejnowski and C.R. Rosenberg, "Parallel networks that learn to pronounce English text", Complex Systems, vol 1, pp 145-168, 1987.
13) Mark Jurik, "Back error propagation - a critique", Proc. IEEE CompCon 88, pp 387-392, 1988.
14) Yoh-Han Pao, "Adaptive Pattern Recognition and Neural Networks", pp 120-140, Addison-Wesley Publishing Co., 1989.
15) R.C. Gonzalez and P. Wintz, "Digital Image Processing", pp 419-425, Addison-Wesley Publishing Co., 1987.
16) Ming-Kuei Hu, "Visual pattern recognition by moment invariants", IRE Trans. Inform. Theory, vol IT-8, pp 179-197, Feb. 1962.