World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:6, 2007

Design and Implementation of a Neural Network for Real-Time Object Tracking

Javed Ahmed, M. N. Jafri, J. Ahmad, and Muhammad I. Khan

Open Science Index, Computer and Information Engineering Vol:1, No:6, 2007 publications.waset.org/5494.pdf

***Abstract*—**Real-time object tracking is a problem which involves extraction of critical information from complex and uncertain image-data. In this paper, we present a comprehensive methodology to design an artificial neural network (ANN) for a real-time object tracking application. The object, which is tracked for the purpose of demonstration, is a specific airplane. However, the proposed ANN can be trained to track any other object of interest. The ANN has been simulated and tested on the training and testing datasets, as well as on a real-time streaming video. The tracking error is analyzed with post-regression analysis tool, which finds the correlation among the calculated coordinates and the correct coordinates of the object in the image. The encouraging results from the computer simulation and analysis show that the proposed ANN architecture is a good candidate solution to a real-time object tracking problem.

***Keywords*—**Image processing, machine vision, neural networks, real-time object tracking.

I. Introduction

REAL-TIME object tracking (OT) is a very specific field of study within the general scope of image processing and analysis. Humans can recognize and track an object perfectly, instantaneously, and effortlessly even in the presence of high clutter, occlusion, and non-linear variations in background, target shape, orientation and size. However, it can be an overwhelming task for a machine! There are partial solutions, but the work is still progressing toward a complete solution to this complex problem.

Tracking of the targets with fixed signatures in stationary backgrounds is a straightforward task for which numerous effective techniques have been developed. When the target signatures or the backgrounds vary in an unlimited or unknown manner, the traditional approaches have not been able to furnish appropriate solutions [1]. Therefore, a new solution based on artificial neural network (ANN) technology is proposed in this paper. The ANN technology provides a number of tools which could form the basis for a potentially fruitful approach to the object tracking problem.

This paper explains the design procedure of an ANN to track an object (here a specific airplane) in Sec. II. The training of the ANN is described in Sec. III. The testing results are illustrated in Sec. IV, and the post-regression analysis is described in Sec. V. Finally, Sec. VI discusses the real-time implementation of the tracking application.

Manuscript received May 20, 2005.

Javed Ahmed is with the Electrical Engineering Department, College of Signals, National University of Sciences & Technology (NUST), Rawalpindi, Pakistan, and also with NESCOM, Islamabad, Pakistan (phone: 92-333-5228285; e-mail: jas123pk@yahoo.com).

M. N. Jafri is with the Electrical Engineering Department, College of Signals, NUST, Rawalpindi, Pakistan (email: mnjafri@yahoo.com).

J. Ahmad is with the Department of Computer Science, Iqra University, Islamabad, Pakistan. (jamilahmad1@hotmail.com).

Muhammad I. Khan is with the Electrical & Automation Division, NESCOM, Islamabad, Pakistan. (mani87_k@hotmail.com).

II. Design of the Neural Network for OT

Design of a neural network involves the selection of its model, architecture, learning algorithm, and activation functions for its neurons according to the need of the application. The objective of our application is to locate a specific airplane in the frames grabbed from a movie clip playing at the speed of 25 frames/second.

A. Selection of the ANN Model

The application, at hand, for which a neural network is to be designed, is a kind of function approximation problem. It may be noted that a back-propagation neural network (BPNN) with one (or more) sigmoid-type hidden layer(s) and a linear output layer can approximate any arbitrary (linear or non-linear) function [2]. The number of hidden layers is normally chosen to be only one to reduce the network complexity, and increase the computational efficiency [3]. Thus, a BPNN is selected for the application at hand, and it consists of three layers: one input layer (of source nodes), one hidden layer (with tangent hyperbolic sigmoid activation function), and one output layer (with pure linear activation function), as shown in Figure 1.

B. Input Layer

The input layer of a neural network is determined from the characteristics of the application inputs. There are 320x240 (i.e. 76800) pixels in each frame coming from a movie (or camera). Each pixel contains three elements (red, green, and blue components). Thus, the total number of elements in a frame is 3x76800 (i.e. 230400). If all these elements are directly put into the neural network, it will be almost impossible to process the image in real-time with a standard PC. Therefore, a preprocessing stage must be incorporated to reduce the size and dimensionality of the input pattern.

Firstly, the color frame is converted into a gray level image, using the following expression for every pixel [4]:

$$y = (0.212671)r + (0.71516)g + (0.072169)b \qquad (1)$$

Open Science Index, Computer and Information Engineering Vol:1, No:6, 2007 publications.waset.org/5494.pdf

where y is the gray level value of the pixel in the output image, and r, g, and b are the red, green, and blue components of the pixel in the input color image, respectively. The values of y, r, g, and b are in the range [0, 255].

Secondly, the gray level image is down-sampled simply by extracting 1st, 5th, 9th, etc. rows and columns, while skipping all other rows and columns in the image. The size of the image is now reduced to 80x60 (reduction factor of 4 with respect to both the number of rows and the number of columns). Thus, the total number of elements is reduced from 230400 to only 4800 with a total reduction factor of 48 (i.e. 3x4x4).

Thirdly, the data of the down-sampled image is normalized, so that the value of each element can be in the range [0.0, 1.0], instead of [0, 255] for fast convergence during the training phase of the ANN. The normalization is done using (2):

$$y_n = y / 255 \tag{2}$$

where y_n is the normalized value.

Finally, the resulting image matrix is reshaped to form a standard pattern (column-vector), by concatenating the rows of the image matrix, and then transposing the large row-vector to make it a 4800-element column-vector. Therefore, the number of input nodes in the proposed BPNN becomes 4800.

C. Hidden Layer

Hidden layer automatically extracts the features of the input pattern [3], and reduces its dimensionality further. There is no definite formula to determine the number of hidden neurons. In this research, a hit-and-trial method was used to identify the number of neurons in the single hidden layer. It was found that only 50 hidden neurons could accomplish the task at hand quite reasonably.

The tangent hyperbolic activation function was chosen for the hidden layer after comparing its converging results with those of the logistic sigmoid function. The tangent hyperbolic function and its fast approximation [5] are given in (3):

$$a_{i1} = \tanh(n_{i1}) = \frac{e^{n_{i1}} - e^{-n_{i1}}}{e^{n_{i1}} + e^{-n_{i1}}} \cong \frac{2}{1 + e^{-2n_{i1}}} - 1 \tag{3}$$

where a_{i1} is i^{th} element of $\mathbf{a}_1$ vector containing the outputs from the hidden neurons, and n_{i1} is i^{th} element of $\mathbf{n}_1$ vector containing net-inputs going into the hidden neurons. $\mathbf{n}_1$ vector is calculated as:

$$\mathbf{n}_1 = \mathbf{W}_{10}\mathbf{p} + \mathbf{b}_1 \tag{4}$$

where $\mathbf{p}$ is the input pattern, $\mathbf{b}_1$ is the vector of bias weights on the hidden neurons, and $\mathbf{W}_{10}$ is the weight matrix between 0^{th} (i.e. input) layer and 1st (i.e. hidden) layer. Each row of $\mathbf{W}_{10}$ contains the synaptic weights of the corresponding hidden neuron.

D. Output Layer

The output layer of the network is designed according to the need of the application output. Since the output of the neural network is expected to produce the row and column coordinates of the target (with respect to the top-left pixel position), the number of output neurons will be two.

Since the frame size is 320x240, the values of the row and column coordinates of the target will be in the ranges [0, 240] and [0, 320], respectively. Thus, the pure linear activation function is selected for the output neurons, and expressed as:

$$\mathbf{a}_2 = \mathbf{n}_2 \tag{5}$$

where $\mathbf{a}_2$ is the column-vector coming from the second output layer, and $\mathbf{n}_2$ is the column-vector containing the net inputs going into the output layer. $\mathbf{n}_2$ is calculated as:

$$\mathbf{n}_2 = \mathbf{W}_{21}\mathbf{a}_1 + \mathbf{b}_2 \tag{6}$$

where $\mathbf{W}_{21}$ is the synaptic weight matrix between the first (i.e. hidden) layer and the second (i.e. output) layer, and $\mathbf{b}_2$ is the column-vector containing the bias inputs of the output neurons. Each row of $\mathbf{W}_{21}$ matrix contains the synaptic weights for the corresponding output neuron.

The designed architecture of the proposed BPNN is shown in Fig. 1. The dimensions of the vectors and matrices are shown under their names, where m_0 (= 4800) is the number of input nodes, m_1 (= 50) is the number of hidden neurons, and m_2 (= 2) is the number of output neurons.

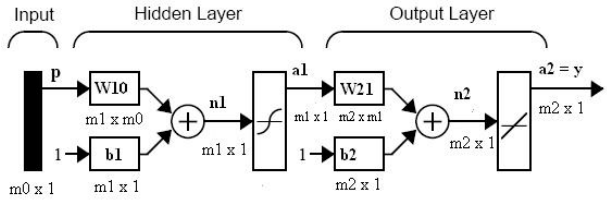

Fig. 1 Architecture of the proposed neural network

III. Training the Neural Network

The training of the neural network was performed using MATLAB and its Neural Network Toolbox. Before training a neural network, a dataset must be prepared on which the network is to be trained. The BPNN is trained in a supervised manner, so the target (desired output) for every training pattern must also be included in the dataset. To generate the dataset, an AVI movie clip of a specific airplane was downloaded from internet [6]. This clip consists of taking-off, flying (with a reasonable variety of maneuvers), and landing profiles of the airplane. It also contains the variations in the background and target illumination and in the scene content.

A user-controlled automatic program was then developed to extract some suitable frames, convert them into valid patterns using the preprocessing stage (as discussed in Sec. II-B), and select the correct coordinates of the object in the image. These coordinates become the elements of the 2-element desired output vector of the proposed ANN. Using this program, a master dataset was generated which contained about 695 examples (pairs of pattern vectors and the corresponding target vectors).

When the network is being trained, it automatically learns the behavior of the I/O mapping, so that whenever those patterns (or similar ones) are shown to it again during its

Open Science Index, Computer and Information Engineering Vol:1, No:6, 2007 publications.waset.org/5494.pdf

application phase, it produces the desirable (or reasonable) outputs.

Another simulation program was then developed to design and train the proposed ANN. This program, initially, divides the training set into two parts: *training set* (to train the network), and *testing set* (to test the performance of the network after training). The *training set* and the *testing set* were chosen to be about $6/7^{th}$ and $1/7^{th}$ parts of the master dataset, respectively. The program, then, simulates the proposed ANN shown in Fig. 1, and trains it in 2491 epochs, using the "Scaled Conjugate Gradient BP Learning Algorithm [7]" with the mean-squared-error goal set to 0.05. Finally, it stores all the synaptic weights of the trained ANN in a binary file. The mean-square-error curve plotted during the training phase is shown in Fig. 2, which shows how the error decreases from about 1050 towards the error goal of 0.05.

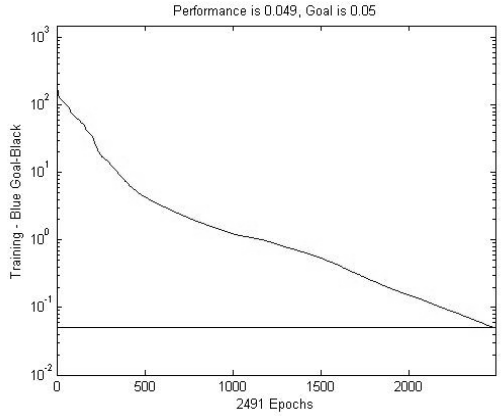

Fig. 1 Mean square error curve of the training

IV. Testing the Neural Network

The network was tested with both the datasets (i.e., *training and testing)*. Fig. 3 shows some of the frames belonging to the *training set.* The target coordinates (suggested by the network, and down-scaled by a factor of 4) are shown at the top of every frame. The center point of the cross-hair sign in every image represents the target coordinates. It can be observed that the network's response is absolutely correct, even in the presence of non-linear variations in background, target shape, orientation and size. Fig. 4 illustrates some of the frames belonging to the *testing set* (which were not shown to the network during its training). This figure shows that the network has learnt the I/O mapping quite well even for the unseen patterns. Fig. 5 shows the result of testing on the frames 3321 to 3332 of the downloaded movie clip.

V. Post-Regression Analysis

The post-regression analysis function compares the actual outputs (A) of the neural network with the corresponding desired outputs (T) [2]. It returns the correlation coefficient (R) between them, and also the slope (m) and the A-intercept (c) of the best-linear-fit equation: $A = mT + c$. The values of m and R can be in the range [0.0, 1.0]. The more the values of m and R are near to 1.0 and the more the value of c is near to zero, the more correct the response of the network. This function also displays a plot of A vs. T. Figures 6(a) and 6 (b) show the result of the analysis on the *row*-coordinates when the network is tested on *training* and *testing sets*, respectively. The same analysis for column-coordinates is shown in Figures 6(c) and 6(d).

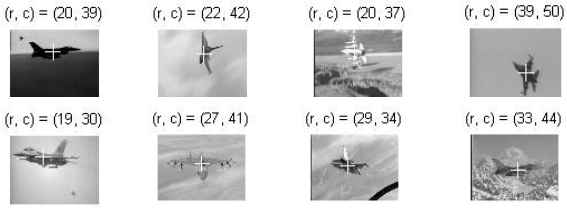

Fig. 2 Some *training set* patterns tested on the network

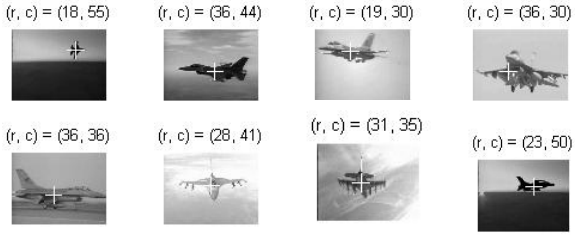

Fig. 3 Some *testing set* patterns tested on the network

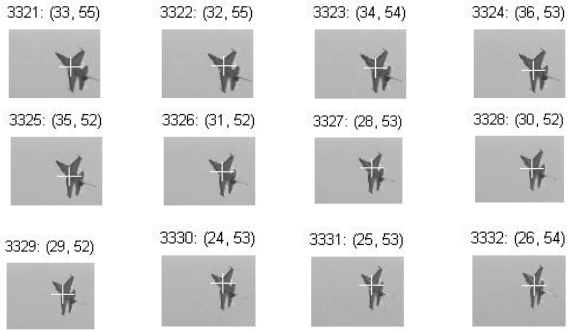

Fig. 4 Network response for a typical video sequence

TABLE I
Summary of the Post-Regression Analysis

Training Set						Testing Set					
Row			Column			Row			Column		
R	*m*	*c*	*R*	*m*	*c*	*R*	*m*	*c*	*R*	*m*	*c*
.999	1	0	1	1	0	.818	.7	8	.745	.7	15

The resulting parameters of the analysis are summarized in Table I. Some figures in the table are rounded to save space. When the network was tested on the *training set*, there was approximately 100% correlation ($R \approx 1$) between the network row-outputs and the correct row-coordinates, and between the network column-outputs and the correct column-coordinates. However, when the network was tested on the unseen *testing set*, there was 81.8% correlation ($R = 0.818$) between the network row-outputs and the correct row-coordinates, and 74.5% correlation between the network column-outputs and the correct column-coordinates. This reduced accuracy for

Open Science Index, Computer and Information Engineering Vol:1, No:6, 2007 publications.waset.org/5494.pdf

unseen patterns is due to the so-called *generalization* problem in neural networks, which can be solved to a significant extent using some techniques described in [2].

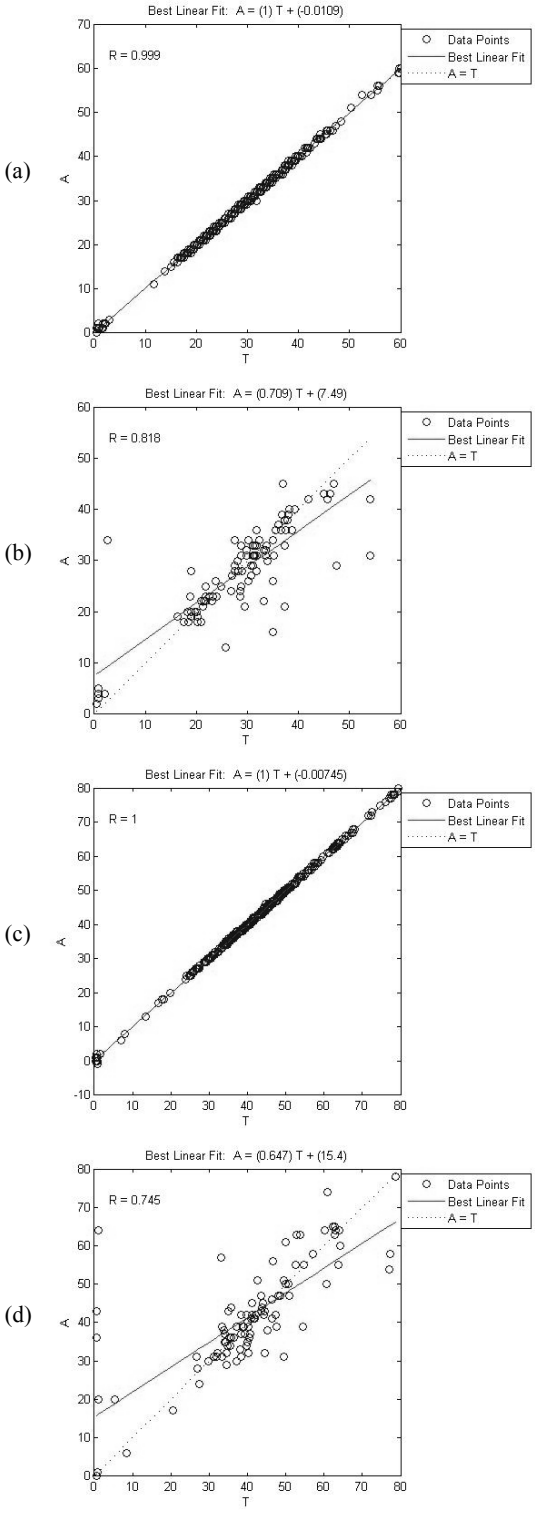

Fig. 6 Post-regression analysis: row-coordinates for (a) training-set, (b) testing-set; column-coordinates for (c) training set, (d) testing set.

VI. Real-Time Implementation

The trained neural network was finally implemented using C++ programming language for the real-time operation. The flow of the application code is outlined in the following steps.

1. Initialize the weight matrices with the parameter values of the network from the binary file (see Sec. III).
2. Grab a frame from the movie clip (or camera).
3. Preprocess the frame (see Sec. II-B) to make a pattern.
4. Present the pattern at the input of the network.
5. The network processes the pattern, and produces the target coordinates.
6. Apply a Gaussian-weighted running average low-pass FIR (finite impulse response) filter on the output values, to make them smooth.
7. Overlay a target sign and coordinates on the image.
8. Show the image with the overlaid content on it.
9. Wait for some milliseconds until next frame's time is due.
10. Go to Step 2.

The program was tested on the downloaded movie clip playing at the speed of 25 frames/second. Thus, the total time available between two frames in the movie was 40 ms. The average time taken by the main steps (i.e. 2 to 8) on two types of PC's is outlined in Table II. Step 5 in the program flow is the actual processing step performed by the neural network. The computational complexity of the proposed BPNN running in application phase is outlined in Table III. Each tanh(.) operation was calculated using its fast approximate formula expressed in (3). The *m*'s are as defined in Sec. II-D.

TABLE II
Average Time Taken by the Main Steps in the Code

PC	Steps 2-8
Centrino Tech., P-IV 2.0 GHz, 2.0 GB RAM	0.03 ms
P-IV 1.5 GHz, 256 MB RAM	11 ms

TABLE III
Computational Complexity of the Neural Network

	Multiplications	Additions	tanh(.)
Input Layer	0	0	0
Hidden Layer	$m_0 m_1$	m_1	m_1
Output Layer	$m_1 m_2$	m_2	0
Total	$m_1(m_0+m_2)$	m_1+m_2	m_1

References

[1] Michael W. Roth, "Survey of Neural Network Technology for Automatic Target Recognition," *IEEE Transactions on Neural Networks*, Vol.1, NO. 1, March, 1990
[2] Howard Demuth, Mark Beale, *Neural Network Toolbox for Use with MATLAB: User's Guide (v. 4)*, The Mathworks, Inc., 2001.
[3] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd Ed., Pearson Education, Delhi, 1999.
[4] *OpenCV: Image Processing and Computer Vision Reference Manual*, http://www.sourceforge.net/projects/opencvlibrary
[5] MATLAB On-line Help Documentation
[6] http://www.fastpasses.com
[7] Moller, M. F., "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525-533, 1993.M. Young, *The Techincal Writers Handbook.* Mill Valley, CA: University Science, 1989.