ARTIFICIAL NEURAL NETWORKS FOR ROBOTICS
COORDINATE TRANSFORMATION

STEPHEN AYLO{1}, LUIS RABELO{2} and SEMA ALPTEKIN{1}

{1}Computer Integrated Manufacturing Laboratory, Engineering Management Department,
223 Engineering Management Building, University of Missouri-Rolla, Rolla, MO 65401 and
{2}Department of Industrial and Systems Engineering, Ohio University, Athens, Ohio 45701, U.S.A.

Abstract—Artificial neural networks with such characteristics as learning, graceful degradation, and speed inherent to parallel distributed architectures might provide a flexible and cost solution to the real time control of robotics systems. In this investigation artificial neural networks are presented for the coordinate transformation mapping of a two-axis robot modeled with Fischertechnik physical modeling components. The results indicate that artificial neural systems could be utilized for practical situations and that extended research in these neural structures could provide adaptive architectures for dynamic robotics control.

1. INTRODUCTION

With the ever increasing complexity of today's manufacturing systems, especially those utilizing the concepts of computer integrated manufacturing (CIM), there is a need for faster and smarter decision making systems.

The introduction of flexible manufacturing cells has increased both the number and importance of industrial robots. However "Traditional approaches to robot control have had limited success in complicated environments, especially for real-time applications" [1]. The main reason for this limited success is that traditional methods use a set of kinematic equations to express the physical environment of the robot. Kinematics work well in the laboratory but are unable to take into account wear and tear of the robots' mechanical and electrical systems. "These systems need high degree of autonomous adaptive learning to be able to...respond to wear and tear of parts effectively" [2]. Traditional approaches are also computationally expensive methods that do not lend themselves to real-time applications.

The conventional method to control the movement of robot arms and end effectors is through the use of kinematics. A system that relates the location of the end effector to the angular movements of the robots' joints. Given the angular movement of each joint and the length of each arm it is a relatively simple procedure to find the coordinate of the end effector. However most of the time the desired end coordinate is known and the objective is to determine the proper joint movements required to arrive at that point. When the robot has multiple degrees of freedom there is no unique solution to this problem as there are more unknowns than equations. This is the inverse kinematics problem and is the most difficult problems in robotics [3]. Kinematics also are limited as the mechanical and electrical systems of a robot are subject to wear and tear over time, therefore the use of kinematics is not suitable in actual use.

These systems need high degree of autonomous adaptive learning to be able to avoid unexpected obstacles and respond to wear and tear of parts effectively. Neural network architectures offer an alternative approach for robot control design [2].

Several papers have been published that have presented possible neural network structures to overcome the weaknesses of kinematics. Roth [4], Elsley [5] and Gardner [6] have each proposed the properties of the backpropagation neural network as ideal for this type problem. Hecht-Nielsen [7] reported on work carried out by James Freeman of Ford Aerospace in Houston, Texas for NASA that used a counterpropagation neural network to guide a robot arm to a gripper fixture on a satellite. The robot arm was assumed to be attached to a repair hanger on the upcoming NASA space station. The network was used to translate an image from a TV camera into distance and angular movements that allowed the robot to grasp the satellite gripper fixture. According to Hecht-Nielsen the entire project from model construction, camera setup, software development
Hecht-Nielson went on to say that additional accuracy could be achieved by replacing the counterpropagation network with the more capable mapping backpropagation network. Additional work by Kuperstein [8] and Saxon and Mukerjee [1] use neural networks linked to machine vision systems for robotic movement control. These three systems add the expense and additional complexity of machine vision systems to the robotic system, limiting their use to special purpose robots. They also present several new problems to be solved involving machine vision.

Josi [9] feels that robotic-control path minimization and collision avoidance will be the next real world application of neural networks. To follow up on this Josi [3] developed a backpropagation neural network to replace the kinematics problem that the calls the most difficult problem in robotics. Josi [3] presented a strong case for the ability of backpropagation neural networks to solve the mathematics involved in coordinate transformation mapping of a two dimension, two jointed robot. Josi used a network that had a total of 36 neurons (processing elements) with two neurons each in the input and output layers and 32 neurons in one hidden layer. The network gradually learns to transform previously unseen input (desired x, y coordinates) into output (joint angles) within his predefined error rate. But he did not address three major questions that need to be answered before real world applications will taken place. These questions and possible solutions to them will be presented.

II. ARTIFICIAL NEURAL NETWORKS: AN OVERVIEW

Neural networks are an effort to emulate some aspects of biological nervous systems, and are capable of spontaneous learning. They can process inexact, fuzzy, ambiguous data that does not exactly match information in memory, unlike a digital computer. Rather than programming a neural network, you "teach" it to provide acceptable answers. As a result, even when you enter new information not previously stored in the network it can still provide adequate responses. This provides neural network the ability to resolve issues that do not have straightforward black and white, yes or no answers. Also, because neural networks work similar to the human brain, they can do well at the type things humans do well and are poor at handling numbers. Accuracy, computational power and logic are not strong points of neural networks [4, 10].

In a neural network, the unit that emulates the biological neuron is called a processing element (see Fig. 1).

The processing element combines, usually by a simple summation, the values of one or more input signals. This combined input is modified by a transfer function. This transfer function can be simple threshold which only produces output if the combined input is greater than the threshold value, or it can be a continuous function which changes the output based on the strength of the combined input. The transfer function produces one output that may be connected to one or more input paths of other processing elements. When connected to other input paths, a connection weight is applied, which corresponds to the synaptic strength of a biological neuron. As each connection has a corresponding weight, the signal on the input path is modified prior to reaching the processing element.

![Fig. 1. A simple processing element.](image)
Fig. 2. A simple neural network.

A neural network is composed of many processing elements joined together as described above. The elements are organized into a sequence of layers that are either fully or randomly connected to successive layers as shown in Fig. 2.

In the operation of a neural network there are two main phases—learning and recall. Neural networks learn in one of three methods: supervised, unsupervised, or self-supervised. Supervised learning is when the programmer provides input and error inputs, teaching the network correct and incorrect responses. In unsupervised learning, data is simply entered and results in internal data clustering. Self-supervised learning occurs when the network uses a feedback loop and corrects errors in the interpretation of data [10, 11].

Recall refers to how the entire network processes its input and creates its output. Many types of networks have an energy function associated with them. In the recall process, each state of the network has an energy value, and the network is iteratively modified until a local minimum in the energy function is found [10].

There are several different learning rules available, among the different rules and procedures developed, it is possible to mention: Backpropagation [12, 17], Counterpropagation [7], Kohonen Feature Maps [13], Adaptive Resonance Theory [14], Hopfield [15], and Restrictive Coulomb Energy [16]. They have their limitations and strengths and it is possible to identify suitable application areas for which they are intended.

In the following sections, the backpropagation algorithm, a supervised training method for feedforward networks, and that is used in this research will be explained.

III. UTILIZATION OF BACKPROPAGATION

This section presents specific reasons for choosing backpropagation for this research [17]. These justification include the ability of backpropagation to model digital (i.e. discrete values of "1" and "0") and analog (i.e. continuous values) data, its inability to memorize data patterns, and its ability to model arbitrary relationships. The following paragraphs will expand upon these features.

A. Modeling digital and analog data

Backpropagation networks can learn how “sharp the threshold should be for each individual input-output relationships” [18]. Consequently, digital logic, analog functions, and even fuzzy predicates can be used to train ANNs.

B. Memorization of data patterns

The complexity of the artificial neural network to be built using backpropagation is determined by the function to be mapped rather than the size of the training set. The middle layer(s) act(s) as a collection of feature detectors [19] and do not memorize the training set. This is true for ANNs which have a reasonable number of hidden units in the middle layer. The output layer can then build an appropriate output pattern based on the particular combination of features detected by the middle layer(s).
C. Modeling arbitrary functions

Backpropagation can be trained to implement a wide collection of relationships, including non-linear ones "with no advanced preprogramming of any internal structure" [18]. The internal structure of the hidden layer(s) developed during the training process models the training data without any advanced information provided.

IV. BACKPROPAGATION LEARNING RULE

The main assumption of backpropagation networks is that all of a network's processing elements are responsible for an erroneous output, unlike some networks (i.e. Hopfield) that blame a single processing element for the state of the network. To place responsibility for an error, the network "propagates" the error backward from the output layer to each previous layer until it reaches the input layer. Typical backpropagation networks are composed of an input layer, an output layer and one or more hidden layers. Figure 3 shows the architecture of a standard backpropagation network.

Before describing the learning rule, an explanation of the notation used is necessary to prevent confusion. A superscript in square brackets is used to indicate which layer of a network is being considered. The remaining notation is as follows:

- $X_{j}^{s}$: the current output, $X$, of the $j$th neuron in layer $s$.
- $W_{ij}^{s}$: the weight, $W$, of the connection joining the $i$th neuron in layer $(s-1)$ to the $j$th neuron in layer $s$.
- $I_{j}^{s}$: the weighted summation of the inputs, $I$, to the $j$th neuron in layer $s$.

In Fig. 4, a standard backpropagation processing element (PE) is shown.

$$I_{j}^{s} = \sum_{i} W_{ij}^{s} X_{i}^{s-1}$$

$$X_{j}^{s} = f(I_{j}^{s})$$
A backpropagation processing element will first sum its inputs then transfer them out as follows:

\[ X^{[l]} = f \left( \sum_j \left( W^{[l]}_{kj} \times X^{[l-1]}_j \right) \right) = f \left( \sum_j \sum_s W_{ks}^{[l]} X^{[l-1]}_j \right) \tag{1} \]

where \( f \) is normally a sigmoid function but can be any differentiable function. The sigmoid function is defined as

\[ f(z) = \frac{1}{1 + e^{-z}} \tag{2} \]

Figure 5 shows two possible transfer functions.

The main feature of backpropagation is how it handles errors. Let \( E \) be the error function. The actual function is not important to understand the process. The critical parameter that is propagated back through the layers is defined by:

\[ e^{[l]} = - \frac{\partial E}{\partial X^{[l]}} \tag{3} \]

Using the chain rule twice gives a relationship between the local error at a particular processing element at level \( s \) and all the local errors at the \( s + 1 \) level which is:

\[ e^{[l]} = f'(x^{[l]}) \times \sum_k \left( e^{[l-1]}_k \times W^{[l-1]}_{kj} \right) \tag{4} \]

If \( f \) is the sigmoid function defined in equation (2), then its derivative can be expressed as a simple function of itself as follows:

\[ f'(z) = f(z) \times (1.0 - f(z)) \tag{5} \]

Therefore from equation (1), equation (4) can be rewritten as

\[ e^{[l]} = X^{[l]} \times \left( 1.0 - X^{[l]} \right) \times \sum_k \left( e^{[l+1]}_k \times W^{[l+1]}_{kj} \right) \tag{6} \]

if its transfer function is sigmoidal. The summation term in equation (6) is used to backpropagate the error in a manner similar to how equation (7) forward propagates the input through the network. In short, a backpropagation network sends the input through the network to the output layer, determines the error and propagates it back through the network to the input layer, where the process is repeated until the error is reduced to a predefined acceptable level.

The aim of the learning process is to minimize, or at least to reduce the global error \( E \) to an acceptable level, by modifying the weights.

Given the current set of weights \( W[s]/l \), we need to know how to increase or decrease them in order to decrease the global error. This can be done using a gradient descent rule as follows:

\[ \Delta W^{[l]}_{kj} = -L \text{ coef} \times \left( \frac{\partial E}{\partial W^{[l]}_{kj}} \right) \tag{7} \]
where $L$ coef is a learning coefficient. The partial derivative in equation (4) can be calculated directly from the local error values, because, by the chain rule and equation (1):

$$
\frac{\partial E}{\partial W^{[l]}} = \left( \frac{\partial E}{\partial I^{[l]}} \right) \times \left( \frac{\partial I^{[l]}}{\partial W^{[l]}} \right) = -e^{[l]} \times X^{[l-1]} \tag{8}
$$

Combining equations (7) and (8) gives:

$$
\Delta W^{[l]} = L \text{coef} \times e^{[l]} \times X^{[l-1]} \tag{9}
$$

So far in this discussion, the existence of a global error function has been assumed. As it is needed to define the local errors at the output layer so they can be sent (propagated) back through the network, it must now be defined. Using the following notation:

$i = \text{input vector}$

$o = \text{output vector (actual)}$

$d = \text{output vector (desired)}$

Then a measure of the global error $E$ can be given by

$$
E = 0.5 \times \sum_{k} (d_{k} - o_{k})^2 \tag{10}
$$

where $k$ indexes the component of the input and output vectors. By equation (3) the local error at each processing element in the output layer can be determined by:

$$
e^{[l]} = -\frac{\partial E}{\partial \theta^{[l]}} = -\frac{\partial E}{\partial o_{k}} = d_{k} - o_{k} \tag{11}
$$

One problem of the gradient descent algorithm is setting the appropriate learning rate. In equation (7) the assumption that the local error is linear does not always hold. At points of high curvature it is important to keep the learning coefficient low to avoid divergent behavior. But a small learning coefficient can lead to very slow learning rates. The concept of a momentum term was introduced to resolve this dichotomy. Equation (9) is modified so that part of the previous delta weight is fed to the current delta weight, acting as a low-pass filter on the delta weight terms.

$$
\Delta W^{[l]} = L \text{coef} \times e^{[l]} \times X^{[l-1]} + \text{momentum} \tag{12}
$$

The introduction of a momentum term reinforces general trends and cancels oscillatory behavior. This allows a low learning coefficient with faster learning.

As stated earlier, the transfer function can be any differentiable function. So far in this discussion a sigmoid function has been used which provides a smooth version of a $\{0, 1\}$ step function. As there are times when a $\{-1, 1\}$ function will more accurately fit the particular problem in question a different function is needed. One acceptable function is the hyperbolic tangent function which is a smooth version of the $\{-1, 1\}$ step function. The hyperbolic tangent function is defined as:

$$
f(z) = (1.0 - e^{-z})/(1.0 + e^{-z}) \tag{13}
$$

As with the sigmoid function, the derivative of the hyperbolic tangent can be expressed in terms of itself:

$$
f'(z) = 0.5 \times (1.0 + f(z)) \times (1.0 - f(z)) \tag{14}
$$

Thus the error propagation equation (4) is modified to:

$$
e^{[l]} = 0.5 \times \left(1.0 + X^{[l]}\right) \times \left(1.0 - X^{[l]}\right) \times \sum_{k} \left(e^{[l+1]} \times W_{kj}^{[l+1]}\right) \tag{15}
$$

Any other smooth function that is differentiable may be selected as a transfer function. Additional examples are not provided here, as it is nearly impossible to define every possible feasible function.
V. PROBLEMS WITH BACKPROPAGATION

There are several problems with the training of backpropagation networks. Two of the most critical are:

A. Finding the proper size of the middle layer

Finding the proper size of the middle layer in a backpropagation network is critical since a middle layer with two few neurons will generate a network which generalizes well but its training performance will not be acceptable. On the other hand, a backpropagation network having a middle layer with too many units will show a deficient behavior due to the tendency that it would have towards “memorizing” it, instead of concentrating on the statistically meaningful patterns that can be extracted from it.

B. Size of the training set

Presenting a good training set is a critical decision. If a very small percentage of the possible patterns are introduced in the training set, the resulting generalization may be poor, while if the opposite is true, the risk to reach local minima or oscillations are higher making it impossible to reach a state of global minima.

To avoid these problems, the utilization and combination of two techniques: Dynamic Node Creation (DNC) and combined Subset Training (CST) was required.

C. Dynamic node creation (DNC)

DNC is an useful tool that helps the search for the optimum size of the hidden layer in backpropagation networks [20]. Fully automated DNC starts with a smaller than ideal number of neurons in the middle layer and then sequentially adds new ones, when needed, according to an algorithm related to the flattening of the average squared error versus training time (iterations). The process stop when this error reaches a user specified value. Unseen testing data is always presented to the network prior to adding a new hidden unit to make sure that accuracy is not being gained at the expense of generalization. As an example, Table 1 and Figs 6 and 7 show the utilization of DNC to search for an optimum network architecture to solve the XOR problem.

D. Combined subset training (CST)

CST, a recent approach [21], that has yielded some good results, proposes introducing a small random subset of all possible input patterns until they are well learned. After this occurs the size of the training set is doubled by introducing another random subset of new input patterns of the same size as the previous one. The network is then retrained on the combined set until it is satisfactorily learned, at which time, the same process of data augmentation is repeated.

VI. APPLICATIONS

The ability of backpropagation's network to learn form training databases has led to numerous applications in several problems. For example, it has been applied successfully in image processing [22], explosives detection [23], sonar targets classification [24] text to speech translation [25] etc.

VII. ROBOT PHYSICAL MODEL

The procedure used to address these weak points was to create a physical simulation of a two-dimensional robot using Fischertechnik robotic modeling components as shown in Fig. 9. The

<table>
<thead>
<tr>
<th>Table 1. XOR problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Fischertechnik robot provided a test platform that was mechanically and electrically unstable. If the neural network could incorporate these faults and provide acceptable solutions then it can provide acceptable solutions for industrial grade robots such as the IBM 7535, which is somewhere between the poor state of the Fischertechnik model and the perfect model that Josin developed.

The Fischertechnik robot is composed of two joints of unequal length. Joint 1 is powered by motor M1 and rotational movement is controlled by potentiometer $EX$. Joint 2 is powered by motor M2 and rotational movement is controlled by potentiometer $EY$. For the simulation the values of $EX$ and $EY$ correspond to one $(X, Y)$ coordinate; however each $(X, Y)$ coordinate not on the outer boundary of the work envelope has two sets of $EX$ and $EY$ values. Figure 10 shows an example of one such point.

VIII. ARTIFICIAL NEURAL NETWORKS FOR ROBOTIC COORDINATE TRANSFORMATION

The coordinate transformation of the robot used in this research may not be a one-to-one mapping. Two configurations, with two different sets of potentiometer values are possible (as it was mentioned above). Therefore, it will be required to utilize a system of two artificial neural networks, each responsible for one of the possible configuration solutions. The mapping assigns a 2-dimensional input vector $(X, Y)$ to a 2-dimensional output vector $(EX, EY)$ containing the
Fig. 7. Application of DNC to the XOR problem.

Fig. 8. Fischertechnik robot model.
desired potentiometer values. A training set with 80 data samples was generated for each possible solution (See Tables 2 and 3).

IX. TRAINING THE ARTIFICIAL NEURAL NETWORKS

A. Training

In order to guarantee the best generalization possible at a given accuracy level, as well as to illustrate the benefits of using a methodical training technique in which both DNC and CST principal are implemented, the original network was set to two input elements, two output elements and only one hidden element. During training nine DNC operations were required, therefore 10 hidden units was the optimal architecture. The CST was applied by increasing the training set from 40 data samples to 80 data samples. The final network was developed and trained in less than four hours on a single 80386 machine with a math coprocessor. Once the first network was trained it was duplicated and trained for the second solution in less than 1/2 h.

Prior to developing the above network, a neural network that was based on the Josin model, but using input from the Fischertechnik robot was developed and trained as a comparison model. This network required 2 input, 2 output and 64 hidden units in 2 hidden layers before it was able to learn. The time and effort required to develop and train this network was 120 h using 5 80286 personal computers with math coprocessors. It should be noted here that this network provided only one of the two possible solutions. The training set for this network was the same as used in the first network described above and is shown in Table 2.
B. Results

In summary, Josin’s work of 1988 was reviewed. Two methods to reduce training time were introduced (DNC and CST). A method to provide a complete set of solutions to the inverse kinematic equations was developed and presented using data from a model robot, from which a set of backpropagation neural networks were developed and trained that (a) dramatically reduced the time required for a development and training of backpropagation networks (b) provided a complete set of solutions to the inverse kinematic equations and (c) were able to take into account the mechanical and electrical faults of the model robot used.

X. CONCLUSIONS

Josin, Saxon and Hecht-Nielsen and others as mentioned in Section II and Section V have proven that neural networks (backpropagation and counter propagation) do in fact have the ability to mathematically solve this problem to within acceptable error rates. It has not been the intent of this work to add additional strength to their claims. It is however intended to provide proof that backpropagation neural networks can provide acceptable solutions to the inverse kinematic problem within an acceptable level of effort on the part of the human developer and provide proof that they are now ready to leave the laboratory and enter the real working world. The network developed in Section V of this paper is capable of taking into account the unknown number of faults, errors and limitations of the poorly constructed Fishertechnik robot and provide solutions that are as accurate as the robot mechanical and electrical systems can support, (i.e. the robot was only capable of plus or minus 0.1 inch repeatability which the neural network had no problem learning.) This ability of the neural network to learn and account for the unknowns present in the
The disadvantage of neural investigations is the high cost of developing and training neural networks, which can be very time-consuming and expensive. However, despite these challenges, neural networks are providing valuable insights into complex systems and can be used to solve a wide range of problems in fields such as robotics and artificial intelligence.

### Table 3. Training set: Network 2

<table>
<thead>
<tr>
<th>Input</th>
<th>Target</th>
<th>Input</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>EX</td>
<td>EY</td>
</tr>
<tr>
<td>1.6000</td>
<td>4.6000</td>
<td>0.3150</td>
<td>0.1170</td>
</tr>
<tr>
<td>0.0000</td>
<td>9.2500</td>
<td>0.3600</td>
<td>0.3800</td>
</tr>
<tr>
<td>9.6000</td>
<td>3.6000</td>
<td>0.3150</td>
<td>0.3900</td>
</tr>
<tr>
<td>7.4500</td>
<td>5.1500</td>
<td>0.3150</td>
<td>0.315</td>
</tr>
<tr>
<td>5.2500</td>
<td>6.5500</td>
<td>0.6750</td>
<td>0.8250</td>
</tr>
<tr>
<td>9.0000</td>
<td>0.0000</td>
<td>0.1950</td>
<td>0.3315</td>
</tr>
<tr>
<td>10.5000</td>
<td>0.0000</td>
<td>0.3150</td>
<td>0.5590</td>
</tr>
<tr>
<td>8.2000</td>
<td>0.0000</td>
<td>0.1650</td>
<td>0.2730</td>
</tr>
<tr>
<td>9.5000</td>
<td>4.5000</td>
<td>0.1950</td>
<td>0.2370</td>
</tr>
<tr>
<td>8.1000</td>
<td>0.5000</td>
<td>0.1800</td>
<td>0.2730</td>
</tr>
<tr>
<td>10.3500</td>
<td>2.0500</td>
<td>0.3600</td>
<td>0.5590</td>
</tr>
<tr>
<td>7.8000</td>
<td>2.5000</td>
<td>0.2400</td>
<td>0.2730</td>
</tr>
<tr>
<td>8.9500</td>
<td>5.6500</td>
<td>0.4350</td>
<td>0.5590</td>
</tr>
<tr>
<td>6.3000</td>
<td>4.5000</td>
<td>0.2850</td>
<td>0.2730</td>
</tr>
<tr>
<td>5.6000</td>
<td>4.6000</td>
<td>0.4200</td>
<td>0.5590</td>
</tr>
<tr>
<td>6.0000</td>
<td>5.6500</td>
<td>0.3150</td>
<td>0.2730</td>
</tr>
<tr>
<td>7.2000</td>
<td>7.7000</td>
<td>0.4950</td>
<td>0.5590</td>
</tr>
<tr>
<td>5.0500</td>
<td>6.5500</td>
<td>0.3600</td>
<td>0.2730</td>
</tr>
<tr>
<td>5.6000</td>
<td>9.0000</td>
<td>0.5550</td>
<td>0.5590</td>
</tr>
<tr>
<td>4.4500</td>
<td>7.0000</td>
<td>0.3750</td>
<td>0.2730</td>
</tr>
<tr>
<td>7.0000</td>
<td>8.0000</td>
<td>0.5100</td>
<td>0.5590</td>
</tr>
<tr>
<td>4.6000</td>
<td>7.3500</td>
<td>0.3900</td>
<td>0.2730</td>
</tr>
<tr>
<td>4.9500</td>
<td>9.4500</td>
<td>0.5850</td>
<td>0.5590</td>
</tr>
<tr>
<td>2.5000</td>
<td>8.0000</td>
<td>0.4350</td>
<td>0.2730</td>
</tr>
<tr>
<td>8.6000</td>
<td>6.2000</td>
<td>0.4500</td>
<td>0.5590</td>
</tr>
<tr>
<td>1.0000</td>
<td>8.3500</td>
<td>0.4800</td>
<td>0.2730</td>
</tr>
<tr>
<td>3.9000</td>
<td>9.9500</td>
<td>0.2850</td>
<td>0.5590</td>
</tr>
<tr>
<td>0.0000</td>
<td>8.4500</td>
<td>0.5100</td>
<td>0.2720</td>
</tr>
<tr>
<td>6.3000</td>
<td>9.6500</td>
<td>0.5250</td>
<td>0.5590</td>
</tr>
<tr>
<td>8.2000</td>
<td>4.5500</td>
<td>0.3150</td>
<td>0.2640</td>
</tr>
<tr>
<td>3.2500</td>
<td>10.2000</td>
<td>0.6150</td>
<td>0.5590</td>
</tr>
<tr>
<td>7.7000</td>
<td>5.4000</td>
<td>0.3375</td>
<td>0.3640</td>
</tr>
<tr>
<td>2.5000</td>
<td>10.4500</td>
<td>0.6300</td>
<td>0.5590</td>
</tr>
<tr>
<td>7.5000</td>
<td>3.1500</td>
<td>0.5400</td>
<td>0.3640</td>
</tr>
<tr>
<td>2.0000</td>
<td>10.4000</td>
<td>0.6250</td>
<td>0.5590</td>
</tr>
<tr>
<td>7.0000</td>
<td>6.3000</td>
<td>0.3600</td>
<td>0.3640</td>
</tr>
<tr>
<td>1.5500</td>
<td>10.6500</td>
<td>0.6450</td>
<td>0.5590</td>
</tr>
<tr>
<td>6.4000</td>
<td>7.0000</td>
<td>0.3900</td>
<td>0.3640</td>
</tr>
<tr>
<td>0.0000</td>
<td>10.7500</td>
<td>0.6750</td>
<td>0.5590</td>
</tr>
<tr>
<td>5.8000</td>
<td>7.5000</td>
<td>0.4050</td>
<td>0.3640</td>
</tr>
</tbody>
</table>

### Table 4. Training set: Network 1

<table>
<thead>
<tr>
<th>Input</th>
<th>Target</th>
<th>Input</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>EX</td>
<td>EY</td>
</tr>
<tr>
<td>10.2500</td>
<td>2.5000</td>
<td>0.3600</td>
<td>0.5590</td>
</tr>
<tr>
<td>5.5000</td>
<td>7.2000</td>
<td>0.3825</td>
<td>0.3380</td>
</tr>
<tr>
<td>9.6000</td>
<td>4.0000</td>
<td>0.3600</td>
<td>0.5005</td>
</tr>
<tr>
<td>5.3000</td>
<td>8.5000</td>
<td>0.4425</td>
<td>0.4020</td>
</tr>
<tr>
<td>8.4000</td>
<td>5.5000</td>
<td>0.2600</td>
<td>0.4230</td>
</tr>
<tr>
<td>3.0000</td>
<td>10.2000</td>
<td>0.5500</td>
<td>0.4810</td>
</tr>
<tr>
<td>9.2000</td>
<td>4.0000</td>
<td>0.3300</td>
<td>0.4290</td>
</tr>
<tr>
<td>3.5000</td>
<td>2.9000</td>
<td>0.1950</td>
<td>0.1170</td>
</tr>
<tr>
<td>5.9000</td>
<td>6.0000</td>
<td>0.3000</td>
<td>0.2860</td>
</tr>
<tr>
<td>4.4000</td>
<td>1.5000</td>
<td>0.3139</td>
<td>0.1300</td>
</tr>
</tbody>
</table>
REFERENCES


