International Journal of Computer Applications (0975 – 8887)
Volume 14– No.8, February 2011

Recognition of Distorted CAD Objects using Neural Networks

Navdeep Kumar
Mechanical Engineering Dept.,
Thapar University, India

Pankaj Garg
Abraham Consultancy Services Pvt.Ltd.,
India

ABSTRACT
The uses of features have been considered to be the technology which bridges the gap between Computer aided design (CAD) and Computer aided manufacturing (CAM) in the CIMS (Computer Integrated manufacturing systems).Active research in the last two decades has resulted in a number of recognition techniques like rule based, graph matching, volume decomposition, hint based, neural network, etc. This paper presents the development of distorted CAD objects recognition system. A well-known multi-layer Perceptron (MLP) neural network with backpropagation learning algorithm is chosen for its fast processing time and its good performance for feature recognition problems. Learning systems that learn from previous experiences and/or provided examples of appropriate behaviors, allow the people to specify what the systems should do for each case, not how systems should act for each step. The recognition system is programmed in C++.

Keywords: CAD, recoginition, artificial neural networs, backpropagation, CAM.

1. CAD FEATURE RECOGNITION
Despite availability of advanced manufacturing and automation technology the link, between CAD and CAM systems, is still not as integrated as desired. The process planning stage, which consists of the explanation of design drawings, is seen as a hindrance in the flow of information between CAD and CAM. An intelligent interface between CAD and CAPP (Computer Aided Process Planning) systems is imperative because the CAPP systems depend on correct data obtained from CAD systems to perform precise process planning. Feature recognition techniques provide such a connection between CAD and CAPP. However, CAD and CAPP systems form different databases. While CAD databases are usually geometry-based, consisting of geometric primitives such as points, lines and arcs, CAPP systems are feature-based such as faces, cylinders, grooves or pockets. It could be said that the CAPP systems describe in terms of manufacturing features, whereas CAD describes parts by their solid model or design features [8].One of the solutions for these problems between CAD and CAM is the automatic feature recognition technique. Miscellaneous techniques including graph-based and hint based such as cell division, cavity volume, convex hull and lamina slicing have been used in automatic feature recognition. There have been many previous attempts to recognize form features for manufacturing purposes, which can be broadly categorized into three areas: Rule-based, Graph based and Neural Network-based systems. One typical example of a rule-based system was developed by Meeran and Pratt using PROLOG. The input to the view orthographic drawings in DXF format. However, the system was limited to prismatic parts and limited by its rules base [3],[4]. Madurai and Lin developed a rule-based system using the expert system approaches for automatic extraction and recognition of part features directly from CAD data for system is three rotational part features. Geometric and topological data of the part in IGES (Initial Graphics Exchange Specifications) format are read by a feature extraction data-compactor, a pre-processor of the system. Manufacturing features are generated by production rules written in LISP. They demonstrated the functioning of the system by two illustrative examples [7]. One important type of a feature recognition method is graph-based recognition, which recognizes features by matching a feature graph to the appropriate subgraph, in a graph representation of the part. This method has advantages such as being applicable to many domains not just machining, allowing the user to add new feature types without changing the codes, being suitable for incremental feature modelling, and being able to recognize isolated features effectively. The disadvantage of this method is that feature interaction and multiple interpretations of features can not be handled well [2]. Dimov [1] proposed a new hybrid method that facilitates the deployment of AFR systems in different application domains. In particular, the method includes two main processing stages: learning and feature recognition. During the learning stage, knowledge acquisition techniques are applied for generating feature-recognition rules and feature hints automatically from training data. Then, these hints and rule bases are utilized in the feature-recognition stage to analyze boundary representation (B-Rep) part models and identify their feature-based internal structure. The proposed AFR method is implemented within a prototype feature-recognition system and its capabilities are verified on two benchmarking parts. But like others this technique is also unable to recognize distorted objects.

In our work Neural network is used in feature recognition because of their capability of learning from examples. The most important advantage of using a neural network is the high possibility of recognizing features. As it is not definite in recognizing some features in the case of rule-based and graph-based systems, the neural network-based systems are able to recognize features that are not certain means distorted sketches.

2. INTRODUCTION TO NEURAL NETWORK
Artificial Neural Network (ANN) is an information-processing paradigm inspired by the way the brain processes information. ANNs are collections of mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. An ANN is composed of a large number of highly interconnected processing elements that are analogous to neurons and are tied together with weighted connections that are analogous to synapses. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Learning typically occurs by example through training or exposure to a truth table set of input/output data where the training algorithm iteratively adjusts the connection weights (Synapses). These connection weights store the knowledge necessary to solve specific problems. The advantages of ANNs lie

in their resilience against distortions in the input data and their capability of learning. They are often good at solving problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found. The basic unit of any neural network is the neuron (processor). Each neuron is able to sum many inputs, whether these inputs are from a database or from other neurons, with each input modified by an adjustable weight as shown in figure 1. The sum of these weighted inputs is added to an adjustable threshold for the neuron and then passed through a modifying (transfer) function that determines the final output. Neurons are arranged in several layers called input, hidden, and output layers .Simply put, the input layer is similar to a matrix of independent variables in a regression while the output layer is the dependent variable. The hidden layer is the series of relationships calculated in the network's training process.

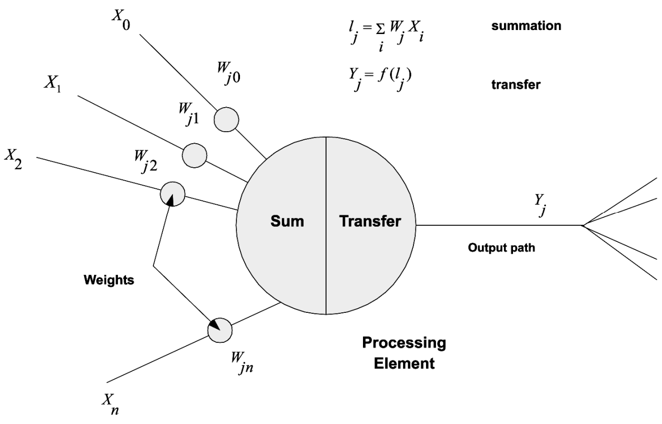

Figure.1: Neuron processing

The idea behind this network is to map CAD object input vectors to CAD object output vectors .Figure 2 shows the typical layout of such a network as it relates to our task of classifying CAD images. In this process mapping occurs by assigning weights to each of the edges. These weights can initially be set to random values, and the neural network will automatically make adjustments to them based on a set of training data. This is the process of the feedforward backpropagation mechanism. Known inputs are fed into the neurons at the input layer, which are then activated and pass the activation information to the hidden layers, which pass their ctivations to other optional hidden layers, and then ultimately the output layer. At this point, the resulting output at the output layer is compared to the desired output. The amount of error at each neuron is then propagated backwards through the network, whereby adjustments to the weights are made accordingly.

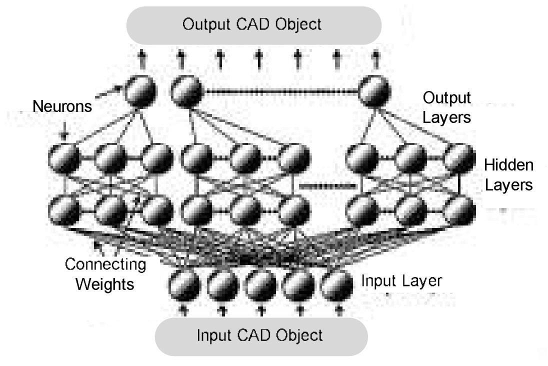

Figure. 2: Main structure of network

3. BACKPROPAGATION ALGORITHM

Back-propagation algorithm is a method of adjusting the output of a multi-layered neural network to produce a desired state for a given input, by first checking the input and computing the required output for that input, then comparing the current output to the required output and adjusting the connection weights to reduce the discrepancy between the required output and the current output, and then repeating this process of adjustment for the next level down in the system and for each lower level of the system in turn down to the lowest level, thus causing the system to learn to produce the required output. The network consists of three distinct layers of units, where each of these units can take on a real numbered value between 0 and 1. The INPUT LAYER, where sets of data are presented to the network, is connected by bi-directional weighted connections to the HIDDEN LAYER which is itself connected by bi-directional weighted connections to the OUTPUT LAYER. All the weights in the network are modifiable, and the network learns to produce the correct input-output mapping by modifying these weights. The backpropagation network is an example of supervised learning, the network is repeatedly presented with sample inputs to the input layer, and the desired activation of the output layer for that sample input is compared with the actual activation of the output layer, and the network learns by adjusting its weights until it has found a set of weights that produce the correct output for every sample input.[9] For example, if we were trying to solve the XOR (exclusive or problem) our training inputs and target outputs would be as in Figure 3.Considering a feedforward network with ***N*** input(i) units, ***L*** hidden(j) units and ***W*** output(k) units, the backpropagation training cycle consists of two distinct phases:

3.1 Forward pass (Refer figure 4):

(A) One of the set of **p** training input patterns is applied to the input layer.

$x_p = (x_{p1}, x_{p2,}... x_{pn})$ which may be a binary or real-numbered vector.

(B) The activations of units in the hidden layer are calculated by taking their net input (the sum of the activations of the input layer units they are connected to multiply by their respective connection weights) and passing it through a transfer function.

$$net_j = \sum_{i=1}^{n} w_{ji} x_i \qquad \text{---- (1.1)}$$

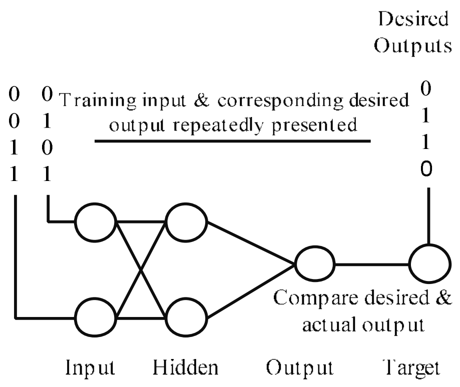
Desired Outputs
0 0
0 1
1 0
1 1
Training input & corresponding desired output repeatedly presented
0
1
1
0
Compare desired & actual output
Input
Hidden
Output
Target

Figure. 3: Training Phase

i) net input to hidden layer unit j
i.e. take the value of each of the n input units connected to it and multiply it by the respective connection weight between them.

ii) output (activation) of hidden layer unit j

$$oh_j = \int (net_j) \quad \text{------(1.2)}$$

i.e. take net input of unit j and pass it through a transfer function

(C) The activations of the hidden layer units calculated in (B) are then used in updating the activation of the output units (or unit in the case of XOR), the activation of the output units is calculated by taking their net input (the sum of the activations of the hidden layer units they are connected to multiplied by their respective connection weights) and passing it through the same transfer function.

i) net input to output unit k

$$net_k = \sum_{j=1}^{L} w_{kj} oh_j \quad \text{---- (1.3)}$$

ii) output of output unit k

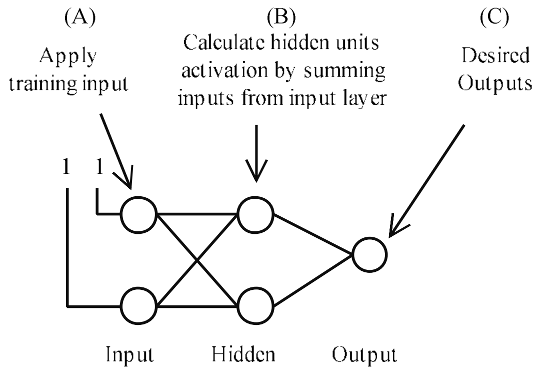
(A)
Apply training input
(B)
Calculate hidden units activation by summing inputs from input layer
(C)
Desired Outputs
1
1
Input
Hidden
Output

Figure 4: Forward Pass

$$oo_j = \int (net_k) \quad \text{------(1.4)}$$

3.2 Backward pass (Refer figures 5 and 6):

(A) The difference between the actual activation of each output unit and the desired target activation (d_k) for that unit is found, and this difference is used to generate an **error signal** for each output unit. A quantity called **delta** is then calculated for all of the output units.

i) error signal for each output unit is difference between its actual output oo_k and its desired output **dk**

$$(d_k - oo_k) \quad \text{---- (1.5)}$$

ii) delta term for each output unit is equal to its error signal multiplied by the output of that unit multiplied by (1 - its output).

$$\delta o_k = (d_k - oo_k) oo_k (1 - oo_k) \quad \text{-- (1.6)}$$

(B) The error signals for the hidden layer units are then calculated by taking the sum of the deltas of the output units a particular hidden unit connects to multiplied by the weight that connects the hidden and output unit. The deltas for each of the hidden layer units are then calculated.

i) error signal for each hidden unit j

$$\sum_{k=1}^{W} \delta o_k w_{kj} \quad \text{..... (1.7)}$$

ii) delta term for each hidden unit j is equal to its error signal multiplied by its output, multiplied by (1 - its output).

$$\delta h_j = (oh_j)(1 - oh_j) \sum_{k=1}^{W} \delta o_k w_{kj} \quad \text{-----(1.8)}$$

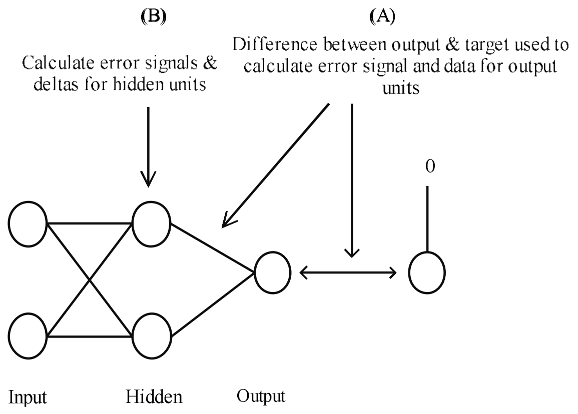
(B)
Calculate error signals & deltas for hidden units
(A)
Difference between output & target used to calculate error signal and data for output units
0
Input
Hidden
Output

Figure. 5: Backward Pass

(C) The **weight error derivatives** for each weight between the hidden and output units are calculated by taking the delta of each output unit and multiplying it by the activation of the hidden unit it connects to. These weight error derivatives are then used to change the weights between the hidden and output layers.

$$wed_{jk} = \delta o_k (oh_j) \quad \text{---- (1.9)}$$

i.e. to calculate the weight error derivative between hidden unit j and output unit k take the delta term of output unit k and multiply it by the output (activation) of hidden unit j

(D) The weight error derivatives for each weight between the input unit i and hidden unit j are calculated by taking the delta of each hidden unit and multiplying it by the activation of the input unit it connects to (i.e. that input pattern x_i). These weight error derivatives are then used to change the weights between the input and hidden layers.

$$wed_{ij} = \delta h_j (x_i) \quad \text{---- (1.10)}$$

To change the actual weights themselves, a learning rate parameter ***n*** is used, which controls the amount the weights are updated during each backpropagation cycle. The weights at a time (t + 1) between the hidden and output layers are set using the weights at a time and the weight error derivatives between the hidden and output layers using the following equation.

$$w_{jk}(t+1) = w_{jk}(t) + \bullet(wed_{jk}) \quad \text{........ (1.11)}$$

In a similar way the weights are changed between the input and hidden units

$$w_{ij}(t+1) = w_{ij}(t) + \bullet(wed_{ij}) \quad \text{--- (1.12)}$$

Using this method, each unit in the network receives an error signal that describes its relative contribution to the total error between the actual output and the target output. Based on the error signal received, the weights connecting the units in different layers are updated. These two passes are repeated many times for different input patterns and their targets, until

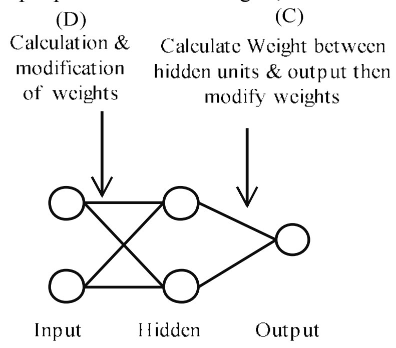
(D)
Calculation & modification of weights
(C)
Calculate Weight between hidden units & output then modify weights
Input
Hidden
Output

Fig. 6: Weight Modification

the error between the actual output of the network and its target output is acceptably small for all of the members of the set of training inputs .This form of training can be applied to much larger networks than the XOR network to solve much more complex problems, but the basic two-pass cycle remains the same. As the network trains, units in the hidden layer organize themselves such that different units learn to recognize different features of the total input space. For example, if a network were trained to respond to a pixel image of the letter 'T', one unit might develop as a feature detector for the vertical bar on the top of the 'T'. After training, when presented with an arbitrary new input pattern which is noisy or incomplete, the units in the hidden layer will respond with an active output if the new input contains a pattern that resembles the feature the individual units learned to recognize during training (for example our unit may still respond to the vertical bar on the top of the 'T' if a pixel were missing). Conversely, hidden-layer units have a tendency to inhibit their outputs if the input pattern does not contain a feature they were trained to recognize (if for example a 'C' were input). These networks tend to develop internal relationships between units so as to organize the training data into classes of patterns (these classes may not be evident to a human observer but can sometimes be detected by applying clustering algorithms to the weights and activations of the units in the network), in this way they develop an internal representation that enables them to generate the desired outputs when given the training inputs, this same internal representation can be applied to inputs that were not used during training, the BPN will classify these new inputs according to the features they share with the training inputs, i.e. these networks have the ability to generalize.

4. WORKING OF DEVELOPED RECOGNITION SYSTEM

The network used for recognition system learns a predefined set of CAD objects as input-output example pairs by using a two-phase propagate-adapt cycle. After a CAD object as input pattern has been passed as a stimulus to the first layer of network units, it is propagated through each upper layer until an output is generated. This output pattern is then compared to the desired output (In our case it is same as input CAD object) and an error signal is computed for each output unit. The error signals are then transmitted backward from the output layer to each node in the intermediate layer that contributes directly to the output. However, each unit in the Intermediate layer receives only a portion of the total error signal, based roughly on the relative contribution of the unit made to the original output. This process repeats, layer by layer, until each node in the network has received an error signal that describes its relative contribution to the total error. Based on the error signal received, connection weights are then updated by each unit to cause the network to converge toward a state that allows all the training patterns to be encoded. The significance of this process is that, as the network trains, the nodes in the intermediate layers organize themselves such that different nodes learn to recognize different CAD objects of the total input space. After training, in the testing phase when presented with an arbitrary CAD object as input pattern that is noisy or incomplete, the units in the hidden layers of the network will respond with an active output if the new input contains a pattern that resembles the CAD object pattern the individual units learned to recognize during training. Conversely, hidden-layer units have a tendency to inhibit their outputs if the input CAD object pattern does not contain the feature that they were trained to recognize. As the signals propagate through the different layers in the network, the activity CAD object pattern present at each upper layer can be thought of as a CAD object pattern with features that can be recognized by units in the subsequent layer. The output CAD object pattern generated can be thought of as a feature map that provides an indication of the presence or absence of many different feature combinations at the input object. The total effect of this behavior is that the BPN provides an effective means of allowing a computer system to examine CAD object patterns that may be incomplete or noisy, and to recognize subtle patterns from the partial input. Using C++ programming tool, a system is developed based on Backpropagation Algorithm to recognize the distorted CAD .In the training phase it demands an input file from user & then by using the input CAD objects it completes its cycle based on algorithm. During the testing phase, when we pass distorted CAD objects it corrects CAD objects output according to its learned knowledge during training phase. As shown in figure.7 for distorted CAD objects line and rectangle the system gives accurate output of objects on the right side.

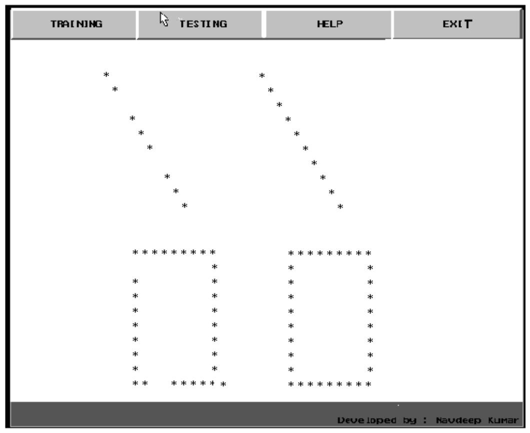

Figure.7: Recognition system

5. CONCLUSION

The significance of this research is in the development of a package for distorted CAD object recognition. It can be concluded that neural networks play significant role in feature recognition system which is the tool to bridge the gap between CAD and CAM in the Computer Integrated manufacturing Environment. Therefore, by integrating the concept of artificial neural networks in CAD\CAM field, we can think to develop intelligent CAD\CAM systems which will be highly interactive.

6. REFERENCES

[1] Dimov, S.-S.; Brousseau, E.-B.: A hybrid method for feature recognition in computer-aided design models, Proceedings of the I MECH E Part B Journal of Engineering Manufacture, 221(1), 79-96

[2] Gao, S.; Shah, J.J.: "Automatic recognition of interacting machining features based on minimalcondition subgraph", Computer Aided Design, 30(9), 727-739 , 1998

[3] Kim, Chongsu.: A representation formalism for feature-based design, Computer-Aided Design, 28(6), 451-460 , 1996.

[4] Meeran, S.; Pratt, M.J.: "Automated recognition from 2D drawings", Computer Aided Design, 25, 7-17, 1993

[5] Rishal, Abu.; Masine, M.-D.: Attribute based feature recognition for machining features, Journal Technology, 46(A), 87-103, 2007.

[6] Shah, J.J.; Mantyla, M.: "Parametric and feature based CAD/CAM", John Wiley & Sons, New York, 1995.

[7] Srinivasakumar, S.M.; Lin, L.: "Rule-based automatic part feature extraction and recognition form CAD data", Computers & Industrial Engineering, 22(1), 49-62 , 1992

[8] Yakup, Yildiz.: Development of a Feature Based CAM System for Rotational Parts ,G.U. Journal of Science 19(1), 35-40 , 2006.

[9] Zurada, M.J.: Introduction to Artificial Neural System, Jacbio Publishing House, Delhi 2008.