SOFTWARE ENGINEERING
CSC 423 B - MWF 11-12

ARTIFICIAL NEURAL NETWORKS
&
BACKPROPAGATION

TO:       Dr. Khaldoun El Khalidi

FROM:     Nadine Ghanem
&
Pedro Maroun Eid

Due: 20 March 2002
ABSTRACT

Also referred to as connectionist architectures, parallel distributed processing, and neuromorphic systems, an artificial neural network (ANN) is an information-processing paradigm inspired by the way the densely interconnected, parallel structure of the mammalian brain processes information. Artificial Neural Networks are computational systems premised upon the principles of biological neural systems; they 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.

TECHNICAL INVESTIGATION

The basic unit of the ANN paradigm is the novel structure of the information processing system. It 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. A processing unit is essentially an equation that is often referred to as a "transfer function" (typically a sigmoid). A processing unit takes weighted signals from other neurons, possibly combines them, transforms them and outputs a numeric result. Processing units are often considered crudely analogous to real neurons and since they are linked together in a network, the name Neural Networks was coined. The basic unit of the BNS is the neuron. In ANS, the basic unit is called various things: neuron, neurode, processing element (PE), and node are popular. A neuron and a PE are not completely analogous. A neuron is a threshold activation device, whereas many nodes employ a threshold function, which scales the PE activation for output. In many cases, this is the sigmoid or logistic function. A PE which uses the sigmoid function is actually more like a collection of neurons in its behaviour -- it acts like a neuronal population where each neuron can have its own, somewhat varying, threshold level.

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 truthed 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.

Although ANNs have been around since the late 1950's, it wasn't until the mid-1980 that algorithms became sophisticated enough for general applications. Today ANNs are being applied to an increasing number of real-world problems of considerable complexity. They are good pattern recognition engines and robust classifiers, with the ability to generalize in making decisions about imprecise input data. They offer ideal solutions to a variety of classification problems such as speech, character and signal recognition, as well as functional prediction and system modelling where the physical processes are not understood or are highly complex. ANNs may also be applied to control problems, where the input variables are measurements used to drive an output actuator, and the network learns the control function. The advantage of ANNs lies in their resilience against distortions in the input data and their capability of learning. They are often good at solving problems that are too complex for conventional technologies (e.g., problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found) and are often well suited to problems that people are good at solving, but for which traditional methods are not.
TOOLS

There are multitudes of different types of ANNs. Some of the more popular include the multi-layer perceptron which is generally trained with the backpropagation of error algorithm, learning vector quantization, radial basis function, Hopfield, and Kohonen, to name a few. Some ANNs are classified as feedforward while others are recurrent (i.e., implement feedback) depending on how data is processed through the network. Another way of classifying ANN types is by their method of learning (or training), as some ANNs employ supervised training while others are referred to as unsupervised or self-organizing. Supervised training is analogous to a student guided by an instructor. Unsupervised algorithms essentially perform clustering of the data into similar groups based on the measured attributes or features serving as inputs to the algorithms. This is analogous to a student who derives the lesson totally on his or her own. ANNs can be implemented in software or in specialized hardware.

Neural networks must 'learn' how to process input before they can be utilized in an application. The process of training a neural network involves adjusting the input weights on each neuron such that the output of the network is consistent with the desired output. This involves the development of a training file, which consists of data for each input node and the correct or desired response for each of the network's output nodes. Once the network is trained, only the input data are provided to the network, which then 'recalls' the response it 'learned' during training.

Backpropagation

The backpropagation paradigm trains a neural network using a gradient descent algorithm in which the mean square error between the network's output and the desired output is minimized. This creates a global cost function that is minimized iteratively by 'backpropagating' the error from the output nodes to the input nodes. Once the network's error has decreased to less than or equal to the specified threshold, the network has converged and is considered to be trained.

A simpler version of backpropagation -- the simple delta rule or the perceptron convergence procedure, which are applicable only in networks with only one modifiable connection layer -- can be proven to find a solution for all input-output mappings which are realizable in that simpler architecture. The error surface in such networks has only one minimum, and the system moves on this error surface towards this minimum (and stays there after it has reached it). This is not true for backpropagation in a network with more than one modifiable connection layers. That is, although in practice one can almost always find a network architecture (even with only two layers of modifiable connections) that can realize a particular input-output mapping, this is not guaranteed. This is because in a network with hidden layer(s) of nodes -- i.e. nodes that are neither in the input nor in the output layer -- the error surface has, in addition to the global, ``lowest'', minimum also local minima, and the system can get stuck in such local error minima.

The theoretical foundations for the derivation of the backpropagation learning rule can be found in Rumelhirt, Hinton, & Williams (1986). Here only a technical description is given.

The following figure shows a typical architecture for networks with backpropagation as the learning rule.
Characteristic is the presence of hidden layers -- only one is shown in the figure -- between the input and the output layers. Also characteristic is that the connectivity structure is feed-forward: that is, there are connections from the input layer nodes to the hidden layer nodes and from the hidden layer nodes to the output layer nodes, but there are no connections backward, for example, from the hidden layer nodes to the input layer nodes. There is also no lateral connectivity within the layers. Also, connectivity between the layers is total in that sense that each input layer node is connected to each hidden layer node and each hidden layer node is connected to each output layer node. Before learning, the weights of these connections are set to small random values. Backpropagation learning proceeds in the following way: an input pattern is chosen from a set of input patterns. This input pattern determines the activations of the input nodes. Setting the activations of the input layer nodes is followed by the activation forward propagation phase: the activation values of first the hidden units and then the output units are computed. This is done by using a sigmoid activation function, like the following one:

\[
a_j = \frac{1}{1 + e^{-\alpha(input_j - \theta)}}
\]

where
- \(a_{j}^{th}\) activation of \(j^{th}\) node in a hidden layer or in the output layer
- \(\text{input}_{ij}\sum_i a_i^j w_{ij}\)
- \(a_{j}^{th}\) activation of \(i^{th}\) node in the previous layer
- \(w_{ij}\) weight of connection from the \(i^{th}\) node in the previous layer, to the \(j^{th}\) node in a hidden layer or in the output layer
- \(\alpha\) constant (determining the steepness of the sigmoid)
- \(\theta\) bias (determining the shift of the sigmoid function along the \``input\'' axis).

The bias is normally realized in backpropagation networks as the part of the input coming from a \``bias node\''. The bias node has an activation of 1 during the whole learning process, and it is connected to each hidden and output layer node. The weights of these connections are changed by learning, just like all other weights. This activation forward propagation phase is followed by the error backward propagation phase. First the error for the output layer nodes is computed by using the following formula:
\[ E_j = (t_j - o_j)a_j(1 - a_j) \]

where
- \([E_j]\) error for node \(j\) of the output layer
- \([t_j]\) "target" activation for node \(j\) of the output layer
- \([o_j]\) actual activation for node \(i\) of the output layer

Then, successively, the error values for all hidden layer nodes are computed:

\[ E_i = a_i(1 - a_i) \sum_j E_j w_{ij} \]

where
- \([E_i]\) error for node \(i\) in a hidden layer,
- \([E_j]\) error for node \(j\) in the layer above,
- \([w_{ij}]\) weight for the connection between node \(i\) in the hidden layer and node \(j\) in the layer above,
- \([a_i]\) activation of node \(i\) in the hidden layer.

At the end of the error backward propagation phase, nodes of the network (except the input layer nodes) will have error values. The error value of a node is used to compute new weights for the connections that lead to the node. Very generally, the weight change is done by using the following formula:

\[ w_{ij} = w_{ij} + \Delta w_{ij} \]

where
- \([w_{ij}]\) weight of the connection between node \(i\) in the previous layer and node \(j\) in the output layer or in a hidden layer,
- \([\Delta w_{ij}]\) weight change for the connection between node \(i\) in the previous layer and node \(j\) in the output layer or in a hidden layer.

The \(\Delta w_{ij}\) values are computed in the same way for each node \(j\) in the network:

\[ \Delta w_{ij} = \beta E_j a_i + m \Delta w_{ij} \]

where
- \([\Delta w_{ij}]\) weight change for the connection between node \(i\) in the previous layer and node \(j\) in the output layer or in a hidden layer,
- \(\beta\) learning rate,
- \([E_j]\) error for node \(j\),
- \([a_i]\) activation for node \(i\) in previous layer, from which the connection originates,
- \([m]\) "momentum" parameter (a value of 0.9 is used in the model),
- \([\Delta w_{ij}]\) for the previous weight change.

The weight change can be done right after the computation of the \(\Delta w_{ij}\) values ("on line" procedure). Alternatively, the \(\Delta w_{ij}\) values can be summed up for all input patterns in the training set, and the actual weight change is done after each input pattern has been presented.
once ("off line" procedure). The backpropagation learning rule is described in Rumelhart, Hinton, & Williams (1986).

APPLICATIONS

Handwritten word recognition and color image segmentation:

Finally, we touch on developments by two teams from the Air Force Institute of Technology (Wright-Patterson Air Force Base, OH).

Gary F. Shartle, Steven K. Rogers, Dennis W. Ruck, Mathew Kabrisky, and Mark O'Hair attacked the problem of handwritten word recognition, stating that, "A machine which can read unconstrained handw Normal + Justified, First line: 0.5"ritten words remains an unsolved problem," reported in their article, "Handwritten word recognition based on Fourier coefficients" (page 290).

For their experiment, they preprocessed four words from a database, written in different hands, by binarizing and cropping. Then Fourier coefficients, computed from the word image, were examined as features for recognition of handwritten words. These features were then processed by a 1 nearest-neighbor or multilayer perceptron classifier. Further, it was found that when the Karhunen-Loève transform was applied to search further for features, the recognition rate was 76.2 percent, the highest of any of the techniques used.

The other team from the Air Force Institute of Technology, Kimberley A. McCrae, Dennis W. Ruck, Steven K. Rogers, and Mark E. Oxley attacked the problem of automated target recognition in their article, "Color image segmentation" (page 306).

Noting that machines do not process images as accurately as humans, but only process intensity information, the authors endeavored to make use of both color and motion cues for more accurate recognition. Machines already have an advantage by being able to process infrared wavelengths as just additional colors.

Using faces, tanks, and airplanes as data sets, the images were color preprocessed. A multilayer perceptron (MLP) performed the color segmentation. The experiment demonstrated that color and motion cues can enhance a computer segmentation system just as they enhance human visual systems.

The authors concluded that color image segmentation is a useful first step in target recognition, and that, even though the neural-network color segmenter gave good results, "this was just a baby step into the color world."