ASSOCIATIVE MEMORY FOR PATTERN RECOGNITION - AN IMPLEMENTATION

ESCADA Jr, J. B.
Instituto de Pesquisas Espaciais
Rod. Pres. Dutra km 40
12.630 Cachoeira Paulista - SP
BRAZIL
Commission II

ABSTRACT

Artificial neural network devices (NND) that can be used as pattern recognizers are discussed here.
NND can be used to store known patterns, in a process known as "learning" or "training", and to map input patterns to the most closely stored pattern (known as "recognition" or "classification").
An implementation of NND as an associative memory in a digital computer, which performs as a classifier for bilevel images is here described. Implementations to allow classification of multi-level (gray) images are also discussed.

1. INTRODUCTION

The neural structure of the human brain has been considered as the basis for the associative structure of the learning process.

A "neuron" may be described as a device with many inputs and a threshold element which weights the many inputs to produce a single output. The most simple neural network model is a collection of "neurons" which interact among themselves, each neuron output driving the inputs of the other neurons. Each connection between two neurons is known as a "sinapse".

A neural network can store patterns as an associative memory, most like an hologram, superimposing the patterns on the same memory medium. When an unknown pattern feeds the inputs of the neural network, the network tends to map the input pattern to the most similar stored pattern.

It is possible, as it is in associative memory, to store pairs of patterns and identifiers (for instance, a number) and the input pattern will then produce, in the output of the neural network, the identifier of the most similar stored pattern. This property of the neural network can be used to implement an image classifier. The advantage of such implementantion is that classification of an input pattern can be achieved in times that are not dependent of the number of the stored patterns. However, if a neural network stores too many patterns, its classification accuracy decreases; there are no formal studies about how much a certain neural network can

"learn", but a few rules may be used to avoid exceeding the network capability.

Use of neural nets should be performed in two steps: the first one consists of "teaching" the network the patterns which will be later compared to an input pattern. This teaching (or training) process can be either adaptive or not. Basically, adaptive training means that the weights of each neuron input are adjusted in order to obtain the best association for that specific pattern. Non-adaptive training implies in a fixed weight for each neuron input.

The second step consists of exciting the neuron network with a pattern and let the neurons network to "reverberate" until its outputs remain stable. When this occurs, the outputs can be read to obtain the stored pattern or, as mentioned above, an identifier of the stored pattern.

2. TYPES OF NEURAL NETWORKS

Here, the most known artificial neural networks are described. Neural nets can be divided in two main classes, according to their capability of storing binary or continuous-valued inputs [1]. Another classification can be achieved if we consider the capability of the neural network for supervised or unsupervised training. However, this study will only deal with supervised training.

HOPFIELD NETWORK

The HOPFIELD network is a special case of the Cohen-Grossberg autoassociator for binary input and is severely limited in the numbers of patterns that can be stored. Also, if two stored patterns have too many similar bits, a misclassification can occur [3].

To implement this network, a correlation matrix should be created to act as the storage medium for all patterns. Suppose the patterns have N bits in lenght, the correlation matrix should be dimensioned as NxN elements matrix. The number of stored patterns M should not exceed INT(0.15 x N x N), provided that two patterns do not have too many similar bits, are not linearly dependent and the patterns to be learned are not degraded by noise.

For example, suppose we want to use this type of network to identify the 26 printed alphabetic characters, each character being defined as an array of 5 x 7 points. The network storage requirement is a matrix of 35x35 elements and the maximum number of patterns which can be stored is M=INT(0.15x35x35) = 183, which is good enough for the latin alphabet.

However, other authors [4,6] have stated that the maximum capacity of the HOPFIELD network is M = N-1. An experimental

implementation of this network showed up that the last statement is more likely to be considered in the example case.

As can be seen, this network does not store any identifier along with the pattern, so the pattern recognition process should have one aditional step: instead of storing the pattern by itself, the product vector of the pattern and a *orthonormal reference* pattern (for instance, the ones provided by the Walsh functions) should be stored; then, after presenting an input pattern, a scalar product of the output pattern by a discriminant function will yield a number which is unique for each learned pattern and can be determined a-priori [7]. This pre-processing of the input patterns also assures that the patterns to be learned are not linearly dependent to each other.

Figure 1 gives the HOPFIELD network algorithm.

BAM NETWORK

Binary bidirectional Associative Memory (BAM) is an adaptation of the HOPFIELD network [3,4]. Pairs (A,B) of patterns of size N and L bits, respectively, are stored and can be later recovered either by presenting pattern A or pattern B to the network. Like the HOPFIELD network, it has severe limitations on the number of patterns that can be stored: the maximum number of pair of patterns which can be stored is M=min(N,L).

The storage requirement of a BAM is a matrix of NxL elements. An implementation of this network showed up that when near its maximum capacity, the BAM becomes very sensitive to noise in the input pattern and can lead to misclassifications.

Figure 2 gives the algorithm for BAM network implementation. It can be observed that when the two input vectors are the same, the BAM network performs as the HOPFIELD network.

HAMMING NETWORK

This network is also used to classify binary patterns. It is composed of two subnets, known as upper and lower subnets. The lower subnet contains NxM synapses, where M is the number of patterns to be stored and N is the number of bits of each pattern. The upper subnet consists of MxM synapses. When an unknown pattern feeds the inputs of the lower subnet neurons, the HAMMING network reverberates until the upper subnet neurons outputs remain stable with only one output being set. So the learning process consists of storing pairs (A,B) in the network, but the B pattern must have only one bit set, the others must remain off.

HAMMING network has one advantage over the two previous ones: it does not suffer from spurious classification [1], but the number of patterns should be known in advance.

The storage requirement for this network is a matrix of MxM elements and a matrix of MxN elements. To store the same 26 alphabetic characters representation of the previous example, the HAMMING network will require 26x26 + 26x35 = 1586 integer numbers.

Implementation of this network showed up that it is not so sensitive to noise as HOPFIELD and BAM networks and needs about the same number of iterations until it converges.

Figure 3 gives the algorithm for HAMMING network implementation.

Of the 3 binary nets described, HAMMING network seems to be the logical choise, in terms of storage requirements (mainly when the number of bits of each pattern is large), reliability and ease of implementation in digital computers.

PERCEPTRON NETWORK

Artificial neural networks for continuous-valued input are restricted to the PERCEPTRON network, which is better described in [2]. It requires more sofisticated training algorithms for weight adjustment, although these algorithms rapidly converge [1]. It also has the advantage of being able to accept several representative patterns to define a "class" and later an unknown input pattern can be associated to one known class instead of being associated to a known pattern. Its operation is very similar to the HAMMING network except that inputs are not binary values and weights must be adjusted during training.

The PERCEPTRON network can be greatly improved if more than one layer of neurons is used, where the outputs of layer *i* are used as input to layer *i+1*. Several studies on this subject demostrate that a three-layer PERCEPTRON is enough to separate almost all classes of patterns, but its storage requirements becomes excessive [1] since the next layer should have at least 3 times the number of neurons of the previous layer for a good performance.

3. IMAGE CLASSIFICATION

An image classification task is in general based on one of the two distinct features of an image: radiometric features, i.e, the gray levels of the pixels of an image, or geometric features, i.e, shapes and ruginess of the image.

RADIOMETRIC CLASSIFICATION

Classifying an image according to its radiometric characteristics has been studied for many years, and several methodologies have been implemented successfully. Use of NND can, however, simplify the implementation of a classifier in

low performance computers, since no excessive arithmetic is needed.

The PERCEPTRON network should be used for supervised classification. The single-layer netowork performs like a Gaussian classifier, and thus it is very restricted for decision regions which are too complex. On the other hand, the three-layer network performs as a k-nearest neighboor classifier and can deal with arbitrary decision regions. However, if the implementation requires the classifier to deal with a large number of classes, the multi-layer network memory requirements can become an inconvenience.

GEOMETRIC CLASSIFICATION

When trying to classify images based on geometric features, the classification task encounters some difficulties:

- patterns are of variable size;

- patterns are most likely to be disturbed by "noise" since they are extracted from real images;

- the images contain bidimensional information which is important for pattern recognition.

The first two constraints can be accomodated with the use of a noise-insensitive classifier. In the first case, a fixed size pattern can be defined and patterns which do not match its size are filled with "blanks", these "blanks" acting as noise to the classifier.

The other constraint (orientation, scaling and alignment of the pattern inside the image may -and normally will- vary between two images) has been subject of several studies and, up to now, there is no conclusive studies for use of a NND to act as a classifier.

When the pattern to be classified is orientation-independent, it is necessary to orient the patterns according to certain rules before storage in the network or before being presented to the network input for classification. This implies the need of computational pre-processing operations for rotation and translation of the patterns.

For certain patterns (for instance, geological structures) the orientation and scaling of the patterns to be classified are significant to the classification process, so the patterns may be presented to the network without any pre-processing except a translation of the image. Even this can be time-consuming since the image should be exactly located to allow the NND to recognize the pattern.

4. CONCLUSIONS

We have seen how artificial neural network devices can be used as pattern recognizers. The time-independence and holographic characteristics of NND used for patterns storage and retrieval make NNDs suitable for implementation in small computers. Several types of NND were briefly discussed and the mass storage requirements for each type were given.

Classifiers implemented with NND can be useful for recognition of noise-disturbed binary images. Radiometric patterns classification can also be easily achieved.

Although NND seems to present a good perspective in the field of geometric patterns classification, the actual state of the art of this subject makes this kind of implementation possible only in fast computers or along with dedicated image-processing hardware devices.

5. BIBLIOGRAPHY

[1] R. P. LIPPMANN, "An Introduction to Computing with Neural Nets", *IEEE ASP Magazine*, April 1987

[2] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press (1969)

[3] B. Kosko, "Bidirectional Associative Memories", *IEEE Transactions on Systems, Man, and Cybernetics*, Fall 1987

[4] B. Kosko, "Constructiong an Associative Memory", *Byte*, September 1987

[5] C. Brown, "Neural Research Yelds Computer That Can Learn", *Electronic Engineering Times*, February 3, 1986

[6] A. Moopenn, J. Lambe, and A. P. Thakoor, "Electronic Implementation of Associative Memory Based on Neural Network Models", *IEEE Transactions on Systems, Man, and Cybernetics*, March/April 1987

[7] Yoh-Han Pao, and F. L. Merat, "Distributed Associative Memory for Patterns", *IEEE Transactions on Systems, Man, and Cybernetics*, November 1975

[8] N. J. Nilsson, *Learning Machines - Foundations of trainable pattern-classifying systems*, McGraw-Hill

Figure 1: Hopfield Network Algorithm

Let X be the known pattern vector;
Let Y be the unknown pattern vector;
Let t be the number of iterations;
Let Z(t) be the output pattern vector at iteration t;
Let M be the the lenght of X.

Step 1: *Convert binary input data to bipolar*

If $X_i = 0$ then let $X_i := -1$, for $1 \leq i \leq M$.

Step 2: *Learn an input pattern*

Let W_{ij} be the connection weight from node i to node j;

$W_{ij} := \text{SUM}\ (X_i \cdot X_j)$ for $1 \leq i \leq M$, $1 \leq j \leq M$ and $i \neq j$.

Step 3: *Initialize for recognition*

$t := 0$;

$Z_i(t) := Y_i$ for $1 \leq i \leq M$.

Step 4: *Reverberate the network*

$t := t + 1$;

For each i, $1 \leq i \leq M$, repeat:

$$Z_i(t) := h\ [\ \text{SUM}\ W_{ij} \cdot Z_i(t-1)\ ],\ \text{for}\ 1 \leq j \leq M$$

where h is the threshold function, implemented as:

$h(s) := -1$ if $s \leq 0$;
$h(s) := +1$ otherwise.

Step 5: *Iterate until convergence*

If $Z_i(t) \neq Z_i(t-1)$ goto step 4, for $1 \leq i \leq M$.

Step 6: *Convert output vector to binary*

If $Z_i(t) = -1$ then let $Z_i(t) := 0$, for $1 \leq i \leq M$.

Figure 2: BAM Network Algorithm

Let X and X' be the two known pattern vector;
Let N and M be the the lenght of X and X', respectively;
Let Y be the unknown pattern vector, of size N or M;
Let t be the number of iterations;
Let Z(t) be the output pattern vector of size M or N
at iteration t.

Step 1: *Convert binary input data to bipolar*

If Xi = O then let Xi := -1, for $1 \leq i \leq N$;

If X'i = O then let X'i := -1, for $1 \leq i \leq M$.

Step 2: *Learn the input patterns X and X'*

Let Wij be the connection weight from node i to node j;

Wij := SUM (Xi·X'j) for $1 \leq i \leq N$, $1 \leq j \leq M$.

Step 3: *Reverberate the network*

t := t + 1;

For each i, $1 \leq i \leq M$ or N , repeat:

Zi(t) := h [SUM Wij·Yi(t-1)], for $1 \leq j \leq N$ or M;

For each i, $1 \leq i \leq N$ or M , repeat:

Yi(t) := h [SUM Wij·Zi(t-1)], for $1 \leq j \leq M$ or N.

Step 4: *Iterate until convergence*

If $Zi(t) \neq Zi(t-1)$ goto step 3, for $1 \leq i \leq M$ or N;

If $Yi(t) \neq Yi(t-1)$ goto step 3, for $1 \leq i \leq N$ or M.

Step 6: *Convert output vector to binary*

If Zi(t) = -1 then let Zi(t) := O, for $1 \leq i \leq M$ or N.

Figure 3: HAMMING Network Algorithm

```
Let X be the known pattern vector;
Let Y be the unknown pattern vector;
Let t be the number of iterations;
Let Z(t) be the output pattern vector at iteration t;
Let N be the the lenght of X;
Let M be the number of patterns to be stored.
```

Step 1: *Convert binary input data to bipolar*

If $X_i = 0$ then let $X_i := -1$, for $1 \leq i \leq N$.

Step 2: *Learn an input pattern*

Let W_{ij} be the connection weight from input i to node j;

$W_{ij} := X_i \div 2$ for $1 \leq i \leq N$, $1 \leq j \leq M$;
$\theta_j := (N-1) \div 2$;

Let W'_{ij} be the connection weight from node i to node j;

$W'_{ij} := 1$ if $i = j$;
$W'_{ij} := \epsilon$ if $i \neq j$ for $1 \leq i \leq M$, $1 \leq j \leq M$;

where $\epsilon > (M-1)^{-1}$.

Step 3: *Initialize for recognition*

$t := 0$;

For each j, $1 \leq i \leq M$, repeat:

$$Z_j(t) := h \left[\sum (W_{ij} \cdot Y_i - \theta_j) \right] \text{ for } 1 \leq i \leq N.$$

Step 4: *Reverberate the network*

$t := t + 1$;

For each j, $1 \leq j \leq M$, repeat:

$$Z_j(t) := h \left[Z_j(t-1) + \epsilon \cdot \sum Z_k(t-1) \right],$$
for $1 \leq k \leq M$ and $k \neq j$;

where h is the threshold function, implemented as:

$h(s) := -1$ if $s \leq 0$;
$h(s) := +1$ otherwise.

Step 5: *Iterate until convergence*

If $Z_j(t) \neq Z_j(t-1)$ goto step 4, for $1 \leq j \leq M$.

Step 6: *Convert output vector to binary*

If $Z_j(t) = -1$ then let $Z_j(t) := 0$, for $1 \leq j \leq M$.

Figure 4: PERCEPTRON Network Algorithm

Let X be the known pattern vector;
Let Y be the desired output vector;
Let t be the number of iterations;
Let Z(t) be the output pattern vector at iteration t;
Let N be the the lenght of X.

Step 1: *Initialize weights*

$t := 0;$

$W_i(t) := \partial_i$, for $1 \leq i \leq N;$

$\theta := \partial;$

where ∂_i are random values.

Step 2: *Determine actual output*

$Z_i(t) := h\ [\ W_i(t) \cdot X_i - \theta\ ].$

Step 3: *Adjust weights*

$t := t + 1;$

$W_i(t) := W_i(t-1) + \Pi \cdot [Y_i - Z_i(t-1)] \cdot X_i$, for $1 \leq i \leq N;$

where $\Pi < 1$, a gain factor.

Step 4: *Iterate until convergence*

If $W_i(t) \neq W_i(t-1)$, for $1 \leq i \leq N$, goto step 3.