Neural Networks Models for Two–dimensional Image Processing and Recognition

Andrzej Dzieliński and Sławomir Skoneczny
Control Division,
Institute of Control & Industrial Electronics,
Warsaw University of Technology,
ul. Koszykowa 75, 00–662 Warszawa, Poland

April, 1993

Abstract

We present different kinds of neural networks for image processing purposes. Multilayer perceptron and Hopfield dynamic network are applied for filtering of images degraded by deterministic kinds of distortions (different types of blurs) and also by stochastic noises of different distributions. Spectral methods as well as spatial algorithms are implemented. A powerful integrated software package as a practical implementation for IBM PC/486 compatibles has been written and tested on several examples.

1 Static Networks–Multilayer Perceptron

Static networks are widely used for pattern recognition/classification and also for image processing [5]. Another area is approximation theory, where the problem may be stated as follows: given two signals $x(t)$ and $y(t)$ find a 'suitable' approximation of the functional relation between the two.

1.1 Static Networks Description

Let us take into considerations a network composed of neurons described by:

$$y_i = \sigma(s_i) + u_i, \quad s_i = \sum_j w_{ij} x_j. \tag{1}$$

A common choice of $\sigma(\cdot)$ is a *sigmoid* function

$$\sigma(s) \to \begin{cases} \to 1, & \text{if } s \to +\infty; \\ \to 0, & \text{if } s \to -\infty, \end{cases} \tag{2}$$

which is usually smooth, but a hardlimiter is also in use. Notice that all signals are denoted as y_i and numbered from top to bottom. Formula (1) uses x_j as inputs to emphasize that the signals fed into static neurons are *not* feedback ones, so not equal to outputs. The reason for introducing the homogeneous notation $y_1, y_2, \ldots, y_N$ will become clear when deriving learning algorithms.

A usual way of arranging the architecture of static networks is to group neurons into *layers* or to design connections in such a way that each of them sends its signal directly only to a certain group of other neurons. The input layer composed of p distribution nodes (not neurons) feeds the first hidden layer, containing n_1 neurons with outputs $y_{p+1}, \ldots, y_{p+n_1}$. This layer, in turn is connected through weights to the second hidden layer of n_2 units and so on until the last layer of q neurons, called the output layer. Notice that each layer l has a direct connection only to the subsequent one, i.e. $l+1$, making the total of L layers. This results in the description

$$y = \sigma(s) + u, \quad s = Wy, \tag{3}$$

where the matrix W is sparse

$$W = \begin{bmatrix} 0 & 0 & \ldots & 0 & 0 \\ W^1 & 0 & \ldots & 0 & 0 \\ 0 & W^2 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 \\ 0 & 0 & \ldots & W^L & 0 \end{bmatrix}. \tag{4}$$

W^i are matrices of interlayer connection weights. The vectors y and u in (3) have the form $y = [y^0\, y^1 \ldots y^L]^T$ and $u = [u^0\, u^1 \ldots u^L]^T$, where y^i, u^i are vectors corresponding to the ith layer; u is the constant bias (offset) vector.

1.2 Backpropagation for Multilayer Perceptron

Backpropagation (BP) was described elsewhere in detail [6], [7]. The reason why it is briefly recalled here is to present it in the rarely used Werbos form [8] Moreover, the essence of the algorithm gives an immense insight into the differences between static and dynamic networks. Last, but not least, useful notation will be introduced.

Define the ordered system of equations

$$y_i = f_i(y_{i-1}, \ldots, y_1), \quad i = 1, \ldots, N+1. \tag{5}$$

Then a systematic and formally proper way of calculating the partial derivatives of y_{N+1} is as follows:

$$\frac{\partial^+ y_{N+1}}{\partial y_i} = \sum_{j>i}^{N+1} \frac{\partial^+ y_{N+1}}{\partial y_j} \frac{\partial f_j}{\partial y_i}, \tag{6}$$

which is a recursive definition of the ordered derivative $\partial^+ y_{N+1}/\partial y_i$, valid only for the systems (5).

The network (3) is given a reference signal d only for the output layer, i.e. $d_i \neq 0$ for $i = N-q, \ldots, N$, and it is required to minimize

$$E = \frac{1}{2} \sum_t (d^t - y^t)^T (d^t - y^t), \tag{7}$$

which is the nonlinear least-squares fitting problem for t patterns. The gradient algorithm for adjusting weights yields

$$w_{ij}^{new} = w_{ij}^{old} - \alpha \frac{\partial E}{\partial w_{ij}}, \quad \alpha > 0, \tag{8}$$

where

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\frac{dy_i}{ds_i}\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\sigma'(s_i)y_j. \tag{9}$$

For the output layer $\partial E/\partial y_i = y_i - d_i$, but it is not straightforward to find this expression for the hidden layers. The backpropagation hypothesis assumes *linear* propagation of the error derivative

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial y_j}\frac{dy_j}{ds_j}\frac{\partial s_j}{\partial y_i} = \sum_j \frac{\partial E}{\partial y_j}\sigma'(s_j)w_{ji}, \tag{10}$$

where y_i belongs to the layer l, and y_j to $l+1$. Starting from the output layer this can be recursively solved. This formulation requires care with neuron indexing and raises doubts about the conformation with the partial derivative definition.

On the other hand, the network structure is ordered (compare (3)–(4) and (5)). Thus, the hypothesis (10) can be expressed as

$$\frac{\partial^+ E}{\partial y_i} = \frac{\partial E}{\partial y_i} + \sum_{j>i}^{N} \frac{\partial^+ E}{\partial y_j}\frac{\partial y_j}{\partial y_i} \tag{11}$$

with E treated as y_{N+1} in (6). Then the BP algorithm for static networks is

$$\begin{aligned} z_i &= \sum_{j>i} w_{ji}\sigma'(s_j)z_j + \epsilon_i, \\ \frac{\partial E}{\partial w_{ij}} &= \sigma'(s_i)z_i y_j, \\ w_{ij}^{new} &= w_{ij}^{old} - \alpha\frac{\partial E}{\partial w_{ij}}, \end{aligned} \tag{12}$$

where $z_i = \partial^+ E/\partial y_i$ and $\epsilon_i = \partial E/\partial y_i$, or in vector-matrix form:

$$\begin{aligned} S &= diag[\sigma'(s_1), \ldots, \sigma'(s_N)], \\ z &= W^T S z + \epsilon, \\ W^{new} &= W^{old} - \alpha S z y^T. \end{aligned} \tag{13}$$

This is inherently an off-line technique as the forward pass must be computed for all t before (13) is applied (backward pass).

2 Hopfield Network and Problem Statement for Associative Memory

There are three processes involved in an associative memory for memorization and retrieval of information :

- storing
- matching
- decision–making

We consider Hopfield network that was suggested by Hopfield in [4] described here by using simple discrete-time autonomous model:

$$\mathbf{v}^{(k+1)} = sgn(\mathbf{T}\mathbf{v}^k), \quad k = 0, 1, \ldots \tag{14}$$

where $\mathbf{v} = [v_1 \ldots v_N]^T$,$v_i = -1$ or $+1$, i=1,...,N
is the input vector,
$\mathbf{T}$ is symmetric $n \times n$ connection matrix,
k denotes instant of time.
The energy function of such network model (neurons with zero threshold) is:

$$E = -\frac{1}{2}\mathbf{v}^T \cdot \mathbf{T} \cdot \mathbf{v} \tag{15}$$

The learning process forms the connection matrix $\mathbf{T}$. The next state of the Hopfield network, i.e., $\mathbf{v}(k+1)$ is computed from the current state by performing the evaluation (14) at a set S of the nodes of the neural network. The modes of operation are determined by the method by which the set S is selected in each time step. If the computation is performed at a single node in any time interval, i.e., $|S| = 1$, then we say that the network is operating in *serial* mode; if $|S| = N$, then we say that the network is operating in a *fully parallel* mode. All the other cases, i.e., $1 < |S| < N$, will be called *parallel* modes.

2.1 Convergence problems

One of the most important problem of the neural network is its convergence. The convergence properties of the model can be summarized as follows:

1. If the Hopfield Network (HN) is operating in a **serial** mode and the connection matrix T is **symmetric** with **zero diagonal**, then the network will always converge to a stable state
2. If HN is operating in a **serial** mode and the connection matrix T is **symmetric** with **nonnegative** elements on the diagonal, then the network will always converge to a stable state
3. If HN is operating in a **fully parallel mode**, then for **arbitrary symmetric** connection matrix T the network will always converge to a stable state or a cycle of length 2; that is the cycles in the state space are of length ≤ 2
4. If HN is operating in a **fully parallel mode**,then for **arbitrary antisymmetric** connection matrix T the network will always converge to a stable state or a cycle of length 4

The second important issue is the number of stable states in the Hopfield network. A state v(k) is called stable (fixed or equilibrium point) iff $\mathbf{v}(k) = sgn(\mathbf{T} \cdot \mathbf{v}(k)) = \mathbf{v}(k+1)$. We want to store only desirable memory (library) vectors in the network but automatically during the learning process (while forming the connection matrix $\mathbf{T}$) we obtain undesirable stable states (spurious states) which is a serious threat during retrieval stage making associative recall more difficult. The number of spurious states is difficult to compute or evaluate and only small part of them can be eliminated [2]. In addition to learning (as defined above), another desirable feature of a neural network is forgetting (i.e. the ability of deleting specified equilibrium points from a given set of

stored equilibria without affecting the rest of the equilibria in a given network). This process called also **unlearning** is examined mathematically to gain some insight into. The information contained in the neural network is stored in the matrix values which represent the connection strength between neurons. It is found that the size of the matrix eigenvalues is very important in determining the frequency of occurrence of a given output state. Unlearning has the property of reducing the eigenvalues of the matrix. Since accessibility is related to the size of the eigenvalues, the states associated with the higher eigenvalues will be reduced more quickly than those with smaller eigenvalues. Thus eigenvalues of stored states will converge to each other, thus making those states more equally accessible. The many spurious states, which each occur infrequently, have eigenvalues which are smaller then those of the desired states. Reducing these eigenvalues further in the unlearning process reduces them to values under unity so that these states decay in the iterative process of recall. The net effect is that the learned states even if they grow more slowly in the iterative process will be more accessible. The unlearning process, since it reduces all eigenvalues will eventually reduce those of the desired states to less than unity so that they will no longer be accessible. This explains why the unlearning process is eventually unstable - too much unlearning destroys the desired memory. But if the unlearning process is controlled carefully it could be a good method of reducing the number of spurious states and could have a stabilizing effect in associative memories [4].

2.2 Energy function learning rules and the connection matrix T

Aiyer *et al.* [1] have found the relationship between the fixed points of the network, energy function and the eigenvalues of the connection matrix $\mathbf{T}$. The learning algorithms we use form **symmetric** T matrix (in order for the Lyapunov energy function to be valid) and because of that $\mathbf{T}$ can be characterized by its eigenvalues and corresponding *orthogonal* eigenvectors denoted as $\lambda_1 \dots \lambda_M$ and $e^1 \dots e^M$. The eigenvalus may be degenerated in which case instead of a corresponding eigenvector there is a corresponding subspace. Also $\mathbf{T}$ may have a degenerate eigenvalue of zero with a corresponding subspace termed the null subspace. The vector $\mathbf{v}$ can be written in terms of its component $\mathbf{v}^i$ in the direction of the eigenvector $\mathbf{e}^i$ plus its component $\mathbf{q}$ in the null subspace as follows:

$$\mathbf{v} = \sum_{i=1}^{M} \mathbf{v}^i + \mathbf{q} \tag{16}$$

where

$$\mathbf{v} \cdot \mathbf{e}^i = \mathbf{v}^i = \gamma \mathbf{e}^i \tag{17}$$

Similarly, the connection matrix, Lyapunov function and dynamic update equation are:

$$\mathbf{T} = \sum_{i=1}^{M} \lambda_i \mathbf{e}^i \mathbf{e}^{iT} \tag{18}$$

$$E = -\frac{1}{2} \sum_{i=1}^{M} \lambda |\mathbf{v}^i|^2 = -\frac{1}{2} \sum_{i=1}^{M} \lambda_j \gamma_i^2 \tag{19}$$

$$\mathbf{u} = \mathbf{T}\mathbf{v} = \sum_{i=1}^{M} \lambda_i \mathbf{v}^i \tag{20}$$

where $\mathbf{u}$ is input vector.

It is obvious that in order to minimize E the network must move $\mathbf{v}$ so as to:

1. Reduce to zero magnitude all $\mathbf{v}^i$'s where $\lambda_i < 0$
2. Increase the magnitude of all $\mathbf{v}^j$'s where $\lambda_j > 0$

If no bounds were placed on the magnitude of $\mathbf{v}$, the network would indefinitely increase $\mathbf{v}$'s magnitude in the direction of the positive λ_j's gradually favoring the larger positive λ_j's. However, as in the Hopfield model, $\mathbf{v}$ is usually limited to the unit hypercube. In this case E is minimized when $\mathbf{v} = \gamma_{max}\mathbf{e}^{max}$ where $\mathbf{e}^{max}$ is the eigenvector corresponding to the largest positive eigenvalue λ_{max}. There is only one positive eigenvalue which is multiply degenerate, then E is minimized when $\mathbf{v}$ lies wholly in the corresponding subspace. Hence, the whole subspace must be considered as the location of the minimum of E.

The way we decide to teach the neural network determines the connection matrix, which stores the memory vectors. The crucial thing in learning algorithm is that the original stored memory vectors must be stable states (or equilibria points) of the network. These stable states should also have some basins of attraction in order that initial states of the network (for example degraded version of memory vector) which lie within this basin may be corrected to the corresponding ideal memory vector.

If $\mathbf{v}$ is stored vector it must be a stable state and the following condition is true

$$v_k = sgn(u_k) = sgn(\sum_{i=1}^{M} \lambda_i v_k^i) \tag{21}$$

So from (16) we have

$$\sum_{i=1}^{M} v_k^i + q_k = sgn(\sum_{i=1}^{M} \lambda_i v_k^i) \tag{22}$$

This can be achieved for any set of memory vectors iff

$\mathbf{q} = 0, \lambda_i = \lambda \quad for \;\; i = 1 \ldots M, \;\; \lambda > 0$

In other words:

1. In order to ensure $\mathbf{q} = 0$ the null subspace must be orthogonal to all memory vectors.
2. So that $\lambda_i = \lambda$ for $i = 1 \ldots M$, the connection matrix $\mathbf{T}$ must have a single positive degenerate eigenvalue corresponding to the memory vector subspace.

3 Conclusions

Two types of neural networks models for image processing and recognition have been applied. A Hopfield network was found to be useful as an associative memory for storing 64 x 64 pixels images in 16 grey levels. The stored images were degraded both by stochastic noises and by deterministic (camera–motion, out–of–focus, atmospheric) blurs. The network showed to be very robust while testing with different types of degradation. The network was able to recognize images even when a human being was not able to do it (up to 49% noised and seriously blurred images were correctly

recognized). However this kind of network could not recognize transformed objects. In order to solve the invariance problems Multilayer Perceptron with BP learning algorithm and FFT algorithm for preprocessing were utilized. A four layers fully connected networks with classical backpropagation algorithm were used. Two types of experiments were performed. In the first an image of size 32 x 32 pixels itself was presented to the network. In the second the FFT of a picture was calculated and only a part of its absolute value (low frequency) was taught and recognized. Thus larger images could be classified (up to 128 x 128 pixels) without lose of accuracy. Although in this network the learning phase takes a long time (48 hours on an i486-based PC) the recognition of shifted pictures was perfect. All simulations were performed on a sequential computer although all neurons can perform in parallel. A hardware realization could significantly improve the performance of presented architectures.

References

[1] S.V.B. Aiyer, M. Niranjan, and F. Fallside. A Theoretical Investigation into the Performance of the Hopfield Model. *IEEE Transactions on Neural Networks*, 1:204–215, 1990.

[2] J. Bruck. *Computing with Networks of Threshold Elements.* PhD dissertation, Stanford University, 1989.

[3] J. Bruck and J.W. Goodman. A Generalized Convergence Theorem for Neural Networks. *IEEE Transactions on Information Theory*, 34:1089–1092, 1988.

[4] J.J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Science*, 79:2554–2558, 1982.

[5] T. Khanna. *Foundations of Neural Networks.* Addison–Wesley Publishing Co., Reading, Massachusetts, 1990.

[6] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1: Foundations.* MIT Press, Cambridge, MA, 1986.

[7] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavior Sciences.* PhD dissertation, Harvard University, Committee on Applied Mathematics, 1974.

[8] P.J. Werbos. Maximizing Long–term Gas Industry Profits in Two Minutes in Lotus Using Neural Network Methods. *IEEE Transactions on System, Men and Cybernetics*, 19:315–333, 1989.