

A neural model for pattern recognition

F. J. López Aligué, I. Acevedo Sotoca, H. González Velasco,
M.A. Jaramillo Morán, M. García Valle

*Departamento de Electrónica e Ing. Electromecánica, Facultad de
Ciencias, Universidad de Extremadura, 06007, Badajoz, Spain*

A neural network model is presented which conjugates the properties of the neuron used and its relationship with the specific connectivity diagram being simulated. The new paradigm is based on the similarity that exists between the configuration of the input to the neuron, which is in its turn a part of the total input to the layer containing that neuron, and the arrangement and value of the weights. Together with this description, a semi-Hebbian learning strategy adapted to this model is presented. The results of a practical implementation to a graphic character recognition process are encouraging for planned further work.

1 Introduction

The design of neural networks for pattern recognition purposes is one of the most significant aspects of work on the structure and functioning of complex systems. Some of these networks can be found in Refs. [1], [2] and [3]. The authors have presented recently a series of theoretical and practical realizations employing neural functions of second and higher order (see López [4] and [5]).

In most of the cases, the networks intended for pattern recognition have a high enough level of dimensionality to allow total connectivity between inputs and processing elements, as in most biological systems. Then, the usual learning strategies may be carried through in more natural ways, such as non-supervised schemes or others of similar characteristics.

In the present work, we show a new formulation for the function of the neuron itself as well as for the learning procedure. The main idea is the correlation between the input configuration of each neuron and its "memory", represented by its weights. The adaptation is carried out according to a method similar to the well-known "winner-take-all", modified to suit our special configuration.

Also, the lateral interaction is in charge of efferent interactions in the way M. Blum [6] suggested some years ago on the basis of a quantity of biological evidence. In the present work, it emerges as the most natural way to implement the rôle of negative (inhibitory) inputs at the same time as the lateral interaction itself.

2 The neural model

Many studies have focused on the use of neural networks to obtain membership functions for many practical applications, as in Kosko [7] and Yager [8]. Our aim was to obtain a classifier made with an array of suitably interconnected single neurons. The solution comes through the definition of a neuron's activity function especially intended for detecting whether the input signal belongs to a certain class. For this purpose, the first step is to postulate such a function as an equivalent of membership functions in fuzzy theory.

396 Artificial Intelligence in Engineering

Taking into account that the usual input vectors and neural weights are of fuzzy nature (with clear biological support), the use of fuzzy functions for the neuron activity function itself seems a plausible approach.

Let

$$X_j = (x_{1j}, x_{2j}, \dots, x_{Nj}) \quad ; \quad x_{ij} \in (0,1) \quad (1.a.)$$

$$W_j = (w_{1j}, w_{2j}, \dots, w_{Nj}) \quad ; \quad w_{ij} \in (0,1) \quad (1.b.)$$

be the input vector to the j -th neuron, and the corresponding weight vector associated with X_j , respectively.

Let us define the function

$$a_{ij} = 1 - |x_{ij} - w_{ij}| \quad ; \quad i = 1, \dots, N \quad ; \quad j = 1, \dots, M \quad (2)$$

and, similarly,

$$a_j = \frac{1}{N} \sum_{i=1}^N a_{ij} = \bar{a}_{ij} \quad (\text{mean value}). \quad (3)$$

As formulated, this last function calibrates the "similarity" existing between the input and the stored pattern, in such a way that, for the special situation where the input coincides exactly with the memorized prototype,

$$a_{ij} = 1 \quad \forall i \Rightarrow a_j = 1,$$

and, in the contrary case, when the input differs absolutely from the prototype (which requires a crisp one or zero value situation),

$$a_{ij} = 0 \quad \forall i \Rightarrow a_j = 0.$$

In normal situations, the function a_j will take on intermediate values as the fuzzy classification suggests. The neuron's activity can thus be formulated as

$$y_j(t) = a_j \{ W_{ij}(t), X_{ij}(t) \} = a_j \{ X_{ij}(t), W_{ij}(t) \}, \quad (4)$$

where we can see that $y_j \in (0,1)$.

In this formulation, in comparison with classical ones, the limiting sigmoid function no longer appears, and the usual sum of x_{ij} by w_{ij} product terms has been replaced by a different, much more easily programmable expression. In addition, the neuron's output is a true measure of the membership function for each input to the class defined by the prototype W_{ij} , as mentioned before. In this sense, this neuron can be seen as a fuzzy neuron, but with some of its properties adapted, namely

$$\sum_{j=1}^M y_j \{ X_i \} \leq M \neq 1, \quad (5)$$

due to the fact that each neuron computes the similarity between the input vector and its own memorized prototype, independently of the rest of the

neurons. That means that each neuron, independently of the others, looks at some properties of the input vector.

Nevertheless, this function remains classical in the sense that it is expressed as an s-norm of t-norms of its variable inputs and weights, and can accept some other expressions if we use any of the following derived definitions, all of them highly significant in fuzzy theory :

$$\hat{a}_j = \text{Max} (a_{ij}), \quad (6.a)$$

$$\circ a_j = 1 - a_j = 1 - \frac{1}{N} \sum_{i=1}^N a_{ij} = \frac{1}{N} \sum_{i=1}^N |x_{ij} - w_{ij}|, \quad (6.b)$$

$$\tilde{a}_j = \text{min} (a_{ij}). \quad (6.c)$$

3 The learning procedure

As a general rule, during the learning phase, the weights must be set according to the prototypes we need to store. For this reason, and taking into account the high number of neurons and prototypes to handle, we propose a non-supervised method as the best choice if one wants to carry out the procedure in the shortest period of time.

Our method makes use of the adaptation of the weights to the intensity of their corresponding inputs by means of a semi-Hebbian model, where the output plays an important rôle in a different form from classical methods.

As usual,

$$w_{ij}(t+1) = w_{ij}(t) + \delta w_{ij}(t+1), \quad (7)$$

where

$$\delta w_{ij}(t+1) = \frac{\bar{x}_{ij}(t+1) - w_{ij}(t)}{\Delta} \quad \text{if } y_j(t) \geq \theta; \quad \theta \in (0,1) \quad (8.a)$$

$$\delta w_{ij}(t+1) = 0 \quad \text{otherwise}, \quad (8.b)$$

and, for programming purposes,

$$\bar{x}_{ij}(t) = \frac{1}{t} \sum_{e=1}^t x_{ij}(t-e), \quad (9.a)$$

which stands for the last "t" intervals. This formulation differs slightly from the purely mathematical one, and must be understood in terms of program implementation needs. In our case, we use its equivalent computer expression

$$\bar{x}_{ij}(t+1) = \frac{\bar{x}_{ij}(t) \cdot t + x_{ij}(t+1)}{t+1}; \quad t = 0, 1, \dots; \quad \bar{x}_{ij}(0) = 0, \quad (9.b)$$

The Δ term is the "learning sensitivity" and offers a way to control the speed of change for each situation. When it is time to adapt the weights quickly (learning phase), Δ is held quite low, allowing the weights to change with the inputs. In

398 Artificial Intelligence in Engineering

the pattern recognition phase, Δ must be set to a high value in order to keep it as constant as the normal use of the network might require. This arrangement is intended to correct only the weights of the neurons that are most "sensitive" to this special configuration of input, namely those that have their output over a predetermined level q , and follows the well-known winner-take-all philosophy. This is particularly interesting when it is impossible to make an all-inputs-to-all-neurons connection, as is the most usual case, for selecting only such neurons that best fit any particular input, leaving the rest unchanged.

4 The lateral interaction

The connectivity shown up to now lacks inhibitory inputs, because all the terms included in eqn (4) are positive. This situation would lead to excitatory-only inputs with a continually growing activity ending in a saturation state no matter what the input is. To correct this, and to allow for some additional functional capabilities, making use of the variants in eqn (6), we insert an "interaction" layer between the inputs of a regular, "processing", layer and the outputs of the previous processing layer, or, if so desired for the first layer, the external input image. The interaction layer is formed from a special kind of neuron, defined by

$$x_h(t) = \hat{a} \{ y_j, y_l \} = |y_j - y_l|, \quad (10)$$

where we can observe that the present outputs constitute the inputs for the next layer, and their inputs come from the outputs of the preceding one. This method for inhibiting one signal by another (the inhibition is total when both signals are equal) can be easily extended to any number of inputs by rewriting eqn (10)

$$x_h(t) = \hat{a} \{ \hat{a} \{ \dots \hat{a} \{ \hat{a} \{ y_1, y_2 \}, y_3 \} \dots \}, y_M \}. \quad (11)$$

5 The network structure

A complete neural network will be implemented by an unlimited sequence of inhibitory and processing layers. If feedback is allowed, some of the outputs of a processing layer will be sent back to a previous inhibitory layer without altering the general schema.

In Figure 1, we can see such an arrangement with some feedback connections shown. There are two processing layers separated by an interaction one. The first receives the incoming image input and processes it. It then sends the results "Y" to the next interaction level, where the lateral inhibition takes place, generating the new inputs "X" for the second processing level. Those inputs enter the next layer for processing, generating the final output of the whole network.

As shown, feedback is only drawn from the second to first layer, but there can be any other inter-layer combination. Also, there can be an interaction layer before the first processing layer. The connectivity diagram, that is, the topological map of intra and inter connections among the neurons, is only limited by the final model used in practice, and determines what part of the total input arriving at the layer will access each neuron, where it will be compared to its weights arrangement (its memory), and the output of this neuron computed according to eqn (4).

6 Practical implementation and application to Pattern Recognition

The whole network has been built on a multiprocessor system developed around a standard bus, the VME. Every CPU board was in charge of the simulation program of a network layer, so giving as many layers as boards. At this date, this work is in its very first stages, but some interesting results are emerging. For our present purposes, we shall present the learning and classification data obtained with a single layer net with 40×40 neurons. This is merely, of course, for showing the performance of the strategy, but it is enough to understand its possibilities.

As we can see in Figure 2, some letters (A, B, C, D, E) as prototypes of their respective classes, along with the outputs obtained, have been used to carry out the learning of the neural network. In the present case, values of 0.6 for the threshold and 20 for the learning sensitivity were chosen, in a learning process consisting of 5 cycles of 100 training sequences for each prototype.



Figure 2. The five prototypes used in the example

At the end of the learning phase, the change of weights is inhibited and some inputs are presented to the network in order to classify them into one of the previously defined classes. Figure 3 depicts the outputs of the network for the Figure 2 inputs, where a white dot indicates those outputs higher than the threshold in the network.

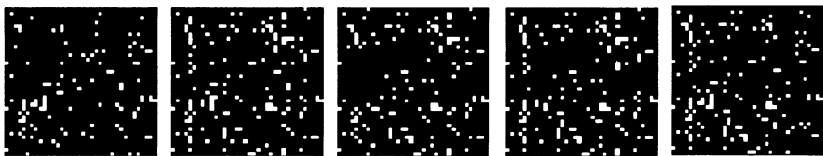


Figure 3. The outputs for the five prototypes

Figure 4 shows the inputs corresponding to three levels, 5%, 20% and 40%, of noise corruption of input "A", and the respective outputs of the network for each input in the recognition phase, that is, with no changes in the weights.

Note that, for clarity and classification purposes, all the outputs have been converted from their analog original values to 0 or 1 binary levels in all the figures shown here (1 = white dot, 0 = black dot), allowing easier pattern recognition. In large networks, those outputs will be addressed to the following layer without any preprocessing, as was noted for figure 1.



Figure 4. The three A noise corrupted (5%, 20%, and 40% respectively) inputs and their corresponding outputs

With those outputs, and using a proprietary nearest neighbor classifier, we got the results shown in Table 1, where the good approximation obtained in all cases is a demonstration of the "descrambler" property of our neural model.

Table 1. The membership values (100% = 1) for each output

	A5%	A20%	A40%
class A	82.000005	75.000000	71.563983
class B	00.000000	02.840909	04.739337
class C	12.000000	10.227273	08.056872
class D	00.568182	00.000000	01.895734
class E	06.000000	11.363637	13.744076

The whole process was implemented on a single VME CPU board, with a learning time of 1.5 seconds for the 5 prototypes (clock frequency of 22 MHz) and less than 1 ms for the recognition phase, including all the input/output processes involved.

7 Conclusions

A new model of neural activity has been developed together with its rules for implementing high density networks. The key is the use of the similarity between the inputs and the weights distribution in the neurons, taking advantage of this "natural" predisposition. Also, a modifiable semi-Hebbian learning strategy hwas provided for the such neural network.

It can be demonstrated that this system tends to a stationary state for any input or connectivity diagram, allowing us to build networks of any complexity. We consider of remarkable interest the fact that the memory is based on the value and the distribution of the weights over the neurons and is strongly dependent on the connectivity diagram once selected, but there it is completely

irrelevant which diagram is initially chosen.

We are, of course, developing soon many more sophisticated networks in much more difficult tasks, but the example presented here has demonstrated the processing capabilities of this neural system. The simulating program, developed in C++, has been generated as a collection of subroutines prepared for use in modular form in the final program, and can be called as needed for each board in the bus, carrying out all the work under the control of a supervisor program for sharing the resources of the system amongst them.

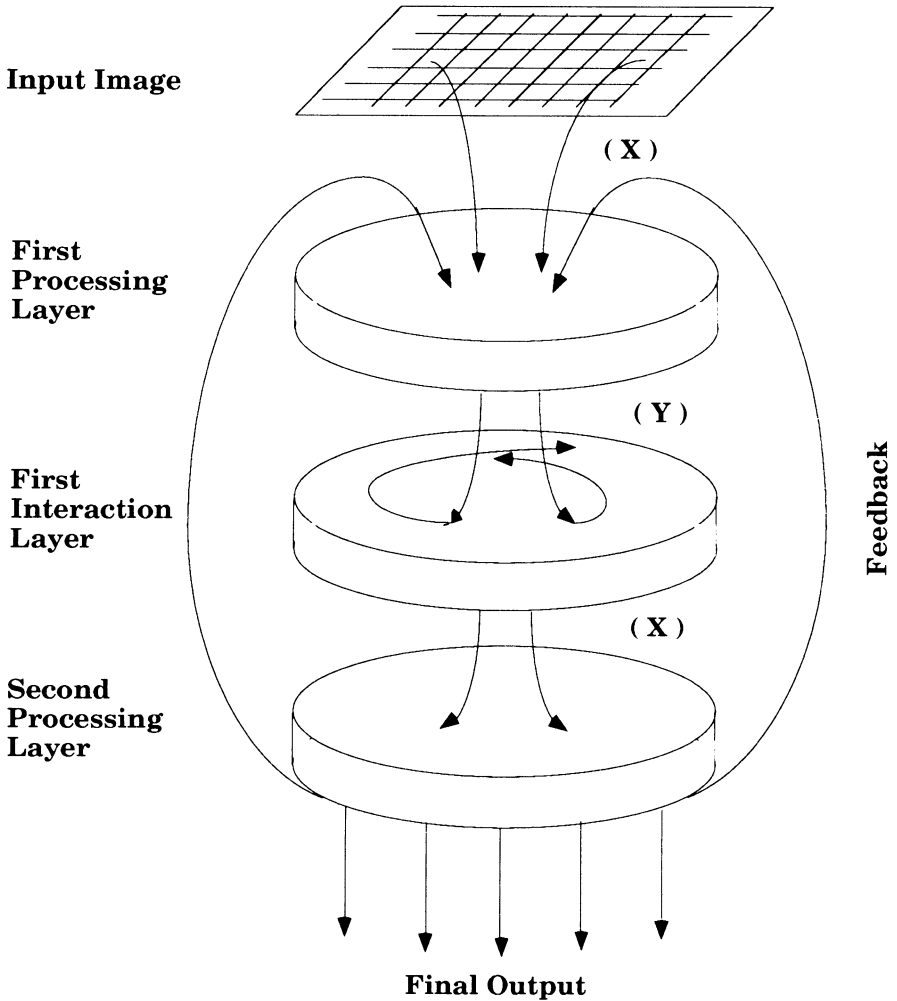


Figure 1. The structure of the network



Acknowledgment

The authors wish to express their gratitude to the Spanish C.I.C.Y.T. for its support to this work through grant TIC 92-0254.

References

- 1.- Chua, L.O. & Yang, L. Cellular neural networks : Theory. *IEEE Trans. on CAS*, 1988, **35**, 10, 1257-1272.
- 2.- Schmidt, W.A.C. & Davis, J.P. Pattern Recognition properties of various features spaces for higher order neural networks. *IEEE Trans. on PAMI*, 1993, **15**, 8, 795-801.
- 3.- Seiler, G., Schuler, A.J. & Nossek, J.A. Design of robust cellular neural networks. *IEEE Trans. on CAS*, 1993, **40**, 5, 358-364.
- 4.- López, F.J., Acevedo, I., Valle, M.G. & Jaramillo, M.A. A general higher order neural model. pp 1295-2003. *Proceedings of the IEEE Intl. Conf. on Neural Networks*, **III**, Orlando, USA, July 1994.
- 5.- López, F.J., Valle, M.G., Acevedo, I. & Jaramillo, M.A. Higher order neuron for cellular image processing, pp 1165-1168. *Proceedings of the ICANN 94*, Sorrento, 1994, Italy.
- 6.- Blum, M. *Properties of a neuron with many inputs*. Principles of Self Organization, New-York: Mac Millan, 1962.
- 7.- Kosko, B. *Neural networks and fuzzy systems*, Englewood Cliffs, NJ: Prentice-Hall, 1991.
- 8.- Yager, R.R. Modeling and formulation fuzzy knowledge bases using neural networks, *Neural Networks*, 1994, **7**, 8, 1273-1283.