A TELEPHONE NUMBER CORRECTOR USING A COUNTERPROPAGATION NETWORK

Juan Moreno [1],Gabriel Sebastián[1], Miguel A. Fernandez[2] & A. Fernandez Caballero[2]
email: [1]{jmoreno,gsebas}@sancho.info-ab.uclm.es [2]{miki,caballer}@info-ab.uclm.es
Departamento de Informática, Escuela Superior Politécnica de Albacete
Universidad de Castilla-La Mancha, 02071 - Albacete, Spain

ABSTRACT

This paper describes the implementation of a system that reminds the user of the telephone number of a given list, even if the user only remembers part of it, or if the given number contains a series of exchanged digits. The system's input consists of a telephone number (composed by digits from 0 to 9), containing from zero up to several generic digits (asterisks, in this case). The system processes the input and returns the selected telephone number among all the learned telephone numbers. In this implementation *auto-associative memories* have been used.

KEYWORDS: Software Tools, Algorithms and Architectures, Memory

1. INTRODUCTION

The following paper introduces the software implementation of a system that, associated to a telephone apparatus, is able to learn a complete telephone agenda. In our system, when the user dials a telephone number, the following cases are accepted: (a) he may dial a complete telephone number, (b) he may dial an incomplete one – where some forgotten numbers are replaced by a generic digit -, (c) he may dial the phone number with some exchanged numbers, (d) he may dial an erroneous number, and, (e) any combination of the previous cases. If necessary, the system corrects the telephone number and returns the most similar telephone number of among all the learned ones. For the implementation of the system we have chosen an auto-associative memory.

Let us suppose that we have L pairs of vectors $\{(x_1,y_1), (x_2,y_2),..., (x_L,y_L)\}$ where $x_i \in \mathbf{R}^n$ and $y_i \in \mathbf{R}^m$. The auto-associative memory presupposes that $y_i=x_i$ and it establishes a correspondence ϕ of **x** in **x** such that $\phi(x_i)=x_i$, and, if some arbitrary **x** is closer to x_i than any other x_j, j=1,...,L, then $\phi(x)=x_i$. For the implementation of the auto-associative memory we have chosen the *counterpropagation network*. Hecht-Nielsen [1] synthesized this architecture combining a structure known as the competitive net with Grossberg's outstar structure [2], [3], [4], obtaining this way the so called counterpropagation networks.

The network's operation is quite simple as shown in figure 1. Given a group of vectors (x_1,y_1), (x_2,y_2), ...,(x_L,y_L), the network learns how to associate a vector X in the input layer with a vector Y in the output layer. If the relationship between X and Y may be defined by means of a continuous function Ω, such that $Y=\Omega(X)$, then the network will be able to learn to approximate that correspondence for all value of X in the interval specified by the training vectors group. In the same way, if the inverse of Ω exists, such that X is a function of Y, then the network will also learn the inverse correspondence, $X=\Omega^{-1}(Y)$[2]. In our special case, we are only interested in the relationship $Y = \Omega(X)$.

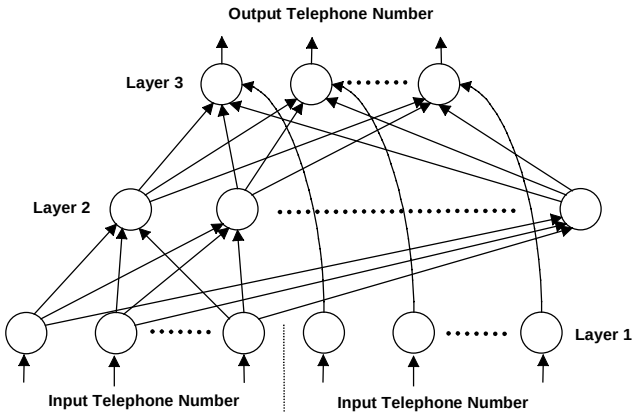

Figure 1: Network structure.

Considering figure 1, note that the architecture consists of three layers. An input vector (see, a telephone number) is applied to the layer 1 units. Each unit of layer 2 calculates its input value *net*, and a competition takes place to see what unit possesses the greatest input value for *net*. This unit is the only one that sends a value to the output units.

2. DESCRIPTION

We have not constructed a hardware implementation of the described system. Our implementation consists of a software simulator programmed under the Visual C++ 5.0 language. Initially, the simulator shows the user a dialogue window where the input telephone number is to be dialed. The user is offered the possibility to dial a known digit (0 to 9), or an asterisk (*) in the place where he doesn't remember the corresponding digit. Once the number has been completely dialed, the network processes the input number and responds with an output number. The system has been trained to learn 100 distinct 9 digit telephone numbers in our latest tests.

Apart from the automatically Visual C++ 5.0 generated classes, we have implemented the classes corresponding to the different layers as well as to the whole counterpropagation network.

The computer network simulators usually impose a normalization for all the input data in order to adapt to the computer CPU calculations. In a counterpropagation network, layer 1 carries out this function. The outputs of layer 1 are governed by the differential equation

$$\dot{x}_i = -Ax_i + (B - x_i)I_i - x_i \sum_{k \neq i} I_k$$

where $0 < x_i(0) < B$ y $A, B > 0$.

Once the input vector X is applied, the process elements rapidly reach a balance state $(\dot{x} = 0)$ [5]. In a digital simulation of the counterpropagation networks, it is possible to simplify the program by software normalizing the input vectors.

That's why, in our simulator, Class *CLayer1* normalizes the input according to the equation $x_i = I_i / \left(\sqrt{\sum_n I_i^2} \right)$.

Layer 2 is the so called competitive network. It is formed by a series of processing elements called *instars* [1], [2]. We suppose that the input vector I and the weight vector *w* have both been normalized. The output of the *instar* is governed by the equation $\dot{y} = -ay + b * net$, where $net = I * w$ and *a,b>0*. The *instar* reaches the balance value in $y^{eq} = (b/a) * net$. The *instar* learns the weight vector w. The learning capacity is carried out initializing with the initial weight vector w, and evolving according to the differential equation $\dot{w} = -cw + dIy$, where y is the output, and c,d>0. The mission of the *instar* is to learn an input vector y, providing a greater output intensity the more it resembles to the learned input vector.

In the implementation of our *CInstar* class, we have given the same values to *a* and *b*. It is possible to simplify the learning of the *instar* by directly assigning the values of the weights of vector *w* from vector y, already normalized. That's the way we have implemented our class *CInstar*. This class also has a parameterized constructor that specifies the number of elements of the vector.

The competitive network consists of a group of *instars* classifying any input vector. The *instar* giving the greatest output value is the winner of the competition, and will be the only one that will have one non null output. The winner will send a value of 1 to the *outstar*, while the rest of *instars* of the competitive network will send the value *0*.

Our class *CCompetitiveNet* incorporates a parameterized constructor, to which the number of *instars* that form the network, corresponding with the number of vectors to be learned, is passed.

Finally, layer 3 consists of some processing elements called *outstars*. During their training process their output values can be calculated by means of

$$\dot{y'}_i = -ay'_i + by_i + c * net_i$$

where the parameters a,b,c>0 and the value of net_i is calculated in the previously described way. Once the training has been accomplished, the output of the *outstar* is equal to:

$$\dot{y'}_i = -ay'_i + cw_i^{eq}$$

where w_i^{eq} it is the weight value that has been obtained during the training session [5], [2]. In a digital simulation, it is possible to approach this learning assigning the weights of the *outstar* directly from the values of the input vectors y.

The *outstar* quickly reaches a balance value equal to the value of the weight encountered in the connection coming from the winner unit of the competitive network. A simple form of visualizing this process consists on realizing that the balance output of the *outstar* is equal to the input *net* of the *outstar*,

$$y'^{eq}_k = \sum_j w_{kj} z_j$$

where z_j is the input received from the corresponding *instar* of the competitive network. Since $z_j = 0$, unless $j=i$:

$$y'^{eq}_k = w_{ki} z_i = w_{ki}$$

This simple algorithm uses balance values of the activities and outputs of the nodes. This way we avoid having to solve numerically all the corresponding differential equations. This way, our class *COutstar* uses the described algorithm to calculate the output. With regard to learning, for each $outstar_i$ of layer 3, our class assigns to each weight w_i the value corresponding to each input vector y_{ij}.

The complete counterpropagation network uses the already described classes *CLayer1*, *CCompetitiveNet* and *COutstar*, forming a general class *CCPN*, with a parameterized constructor incorporating two parameters that indicate the number of *instars* and the number of elements of the input vectors. That is to say, the class *CCPN* can contain a varied number of vectors to learn and of elements of these vectors. This way, this design allows us to implement flexible architectures of counterpropagation networks.

The outline of our counterpropagation network is the following one:

- Layer 1 or input layer consists of 9 nodes in the input vector X, corresponding to the number of digits of the telephone number marked by the user. With regard to the y inputs, we have 900 nodes

corresponding to the 9 digits of the 100 telephone numbers to learn.

- The hidden layer, the competitive network, is formed by 100 *instar* nodes, where each *instar* learns one telephone number.

- Finally, layer 3 or output layer, consists of 9 *outstars*, each one memorizing its corresponding digit of the y input vectors. The output of this layer will be the selected telephone number.

We have chosen the counterpropagation network to implement this system, due to the fact that combining already existing types of networks in a new architecture, offers the possibility to form different and useful networks starting from the existing structures. The counterpropagation networks use different learning algorithms for each layer, allowing the network to be very quickly trained. As a counterpart these networks don't always offer enough precision for some applications. In our special case, looking at the obtained results, it is valid for our system.

With regard to the chosen parameters, let's comment that we have chosen 9 component input vectors, as in our country (Spain), the phone numbers consist of 9 digits. We consider that a 100 telephone number agenda is a sufficiently significant sample as to check the validity of this system's counterpropagation network. Consider that these 100 telephone numbers have been chosen randomly.

As it has already been commented, the user marks the digits that he doesn't remember with an asterisk (*). Each input digit generates the intensity shown in table 1.

Digit	Intensity
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
*****	0 or 5

Table 1: Input intensities for each digit.

As it is possible to observe in table 1, the input intensity *i+1* corresponds to each $digit_i$, since the input intensity, having been normalized, doesn't influence too much in the obtained results. In the case of digit *, the best input intensity is *5*, since it is the one that best approaches to the intensity of the rest of the digits. Better results are obtained using this intensity rather than other intensities like the null one, as it may be observed in the results section.

3. RESULTS

For the verification of the results we have selected a sample of 10 telephone numbers. With this sample we have studied the behavior of the network under the following cases: the telephone number correctly dialed, the telephone number containing one, two and three *, the telephone number dialed with one and two erroneous digits, and finally, the telephone number exchanging two digits.

In the following table (table 2), the validity of the system may be graphically observed:

Telephone	C	*	**	***	C1	C2	I2
967212356	✔	✚/❑	❑	❑	✔	✔	✔
921456753	✔	✚/❑	❑	❑	✔	✔	✔
999876543	✔	❑				✔	✔
967534389	✔	✚/❑			✔	✔	✔
934666543	✔	❑	❑		✔	✔	✔
965436745	✔	❑	❑		✔		✔
945730980	✔		✚			✔	
933458658	✔	❑	❑		✔		✔
978785656	✔	❑			✔		✔
965667933	✔	✚/❑	❑	❑	✔	✔	✔

Table 2: Results of the simulation.

- **C**: Behavior of the network dialing the telephone number correctly.

- ***: Behavior of the network dialing the telephone number with a generic digit.

- ****: Behavior of the net dialing the telephone number with two generic digits.

- *****: Behavior of the network dialing the telephone number with three generic digits.

- **C1**: Behavior of the network dialing the telephone number with an erroneous digit.

- **C2**: Behavior of the network dialing the telephone number with two erroneous digits.

- **I2**: Behavior of the network dialing the telephone number with two exchanged digits.

- ✔: The network responds with the correct telephone number, the most similar of among those learned.

- ✚: The network responds with the correct telephone number, when we assign to the generic digits an intensity of *0*.

- ❑: The network responds with the correct telephone number, when we assign to the generic digits an intensity of *5*.

- An empty cell expresses that the network responds with an incorrect telephone number.

It is easy to observe the improvement of results that is offered by assigning to the generic digits * an intensity of 5, with regard to those obtained assigning them the intensity of 0. It is also deduced that the behavior of the network is quite robust except for the case in which the user dials a telephone number with three generic digits. Finally, it is necessary to highlight the excellent behavior appreciated in the case of the exchange of two digits.

4. CONCLUSIONS

The counterpropagation network is a good solution for the system that is sought to be built in hardware, providing an excellent behavior in the case of assigning a value of 5 to the intensity of the generic digit. It also incorporates a quick and simple learning mechanism. You have only to assign the telephone number digits to the weights. And the answer is obtained in a quick way.

References

[1] Hecht-Nielsen R., *Neurocomputing,* Addison-Wesley, Reading, MA, 1990.

[2] Grossberg S., "Studies of Mind and Brain", *Boston Studies in the Philosophy of Science,* **vol. 70**, D. Reidel Publishing Company, Boston (1982).

[3] Hecht-Nielsen R., "Counterpropagation networks", *Applied Optics*, **vol. 26, no. 23,** pp. 4979-4984 (1987).

[4] Hecht-Nielsen R., "Counterpropagation networks", Proceedings of the IEEE First Internacional Conference on Neural Netwoks, Piscataway, NJ, pp. II-19 - II-32, IEEE (1987).

[5] Freeman J.A. & Skapura D.M., *Neural Networks. Algorithms, Applications, and Programming Techniques,* Addison-Wesley, Reading, MA, 1991.