A Neural Architecture for the Identification of Number Sequences

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

Abstract

This paper describes an architecture based on spatio-temporal networks that identifies sequences of numbers. This architecture incorporates an input layer that transforms (by means of a mathematical function) the system's input into a normalized vector that will be applied in a second step to a spatio-temporal network. Finally, the architecture is completed by an output layer using Grossberg's outstar units [1]. We have appreciated our system's complexity to be lower than any other existent method developed to solve problems of this type. By means of this architecture we have implemented a system that reminds a 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 processes the input and returns the selected telephone number among all the learned ones.

1. Introduction

The following work describes an architecture based on *spatio-temporal networks* to identify sequences of numbers. The architecture incorporates an *input layer* whose output is a normalized vector obtained according to a mathematical expression that will be explained in next section. The architecture also incorporates a *hidden layer* that receives the output of the former *input layer*. This *hidden layer* is a *spatio-temporal network* (*STN*). Finally there is an *output layer* (*outstar*) whose function is to choose the output of among all the previously learned ones.

This architecture has inspired a system that, incorporated to a telephone apparatus, memorizes an agenda of telephone numbers. When the user dials a telephone number, either a complete one, or where some digits are missing, or even where some digits have been exchanged, the system returns the correct telephone number (obviously if it was erroneous).

We will use a memory in the following terms. Let's have L pairs of vectors $\{(x_1,y_1), (x_2,y_2),..., (x_L,y_L)\}$ where $x_i \in \boldsymbol{R}^n$ and $y_i \in \boldsymbol{R}^m$. This memory presupposes that $y_i = x_i$; establishing this way a correspondence ϕ from x into x such that $\phi(x_i)=x_i$, and, if any arbitrary x is closer to x_i than any other x_j, where $j=1,... , L$, then $\phi(x)=x_i$. For the implementation of this memory we have chosen the idea of *spatio-temporal networks* .

In *spatio-temporal networks*, the input vectors that are presented to them are correlated in time. Their architecture is based on the structure of formal *avalanche* proposed by Grossberg [2]. Instead of carrying out a recognition operation, the *avalanche* allows to learn and to remember a sequence of *spatio-temporal patterns*.

In our particular application each dialed digit corresponds to one normalized vector; the sequence of digits through the time will form the final telephone number.

If $\{Q_{11}, Q_{12}, ..., Q_{1n}\}$ is defined as the sequence of input digits to our network, the *avalanche* for the recognition of a telephone number is the one represented in figure 1.

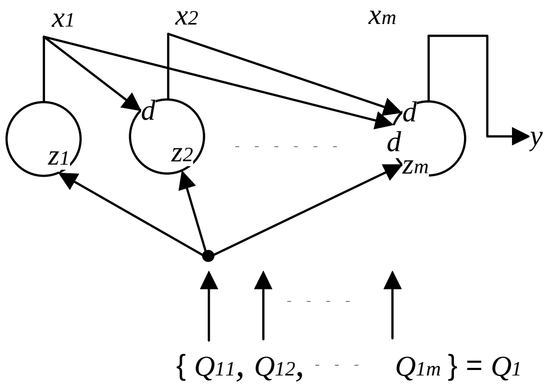

Figure 1. The avalanche network structure.

Every Q_{1i} is transformed into a normalized vector by means of the function defined in the *input layer*, and is applied to the inputs of all the units of the *STN*. These inputs are allowed to remain there during a time *t*. Later the following vector Q_{1i+1} is applied, and so forth.

During that time *t*, each unit dynamically adjusts its activation and output values according to a rule that will be introduced later. Observe that the output of each unit is connected to all successive units, from left to right, with a connection intensity *d*, where $0 < d < 1$.

After processing all the digits in the *input* and *STN layer*, the *outstar* chooses one specific telephone number of among all the learned ones.

2. Description

Based upon the above ideas, we have designed an architecture consisting of three layers, as shown in figure 2. We may distinguish an *input layer* with one single processing element that transforms the input digit. There is also a *hidden layer* based on a *spatio-temporal network* that is able to learn a sequence of normalized vectors. Finally consider also the *output layer* that associates the winning sub-network to one particular telephone number.

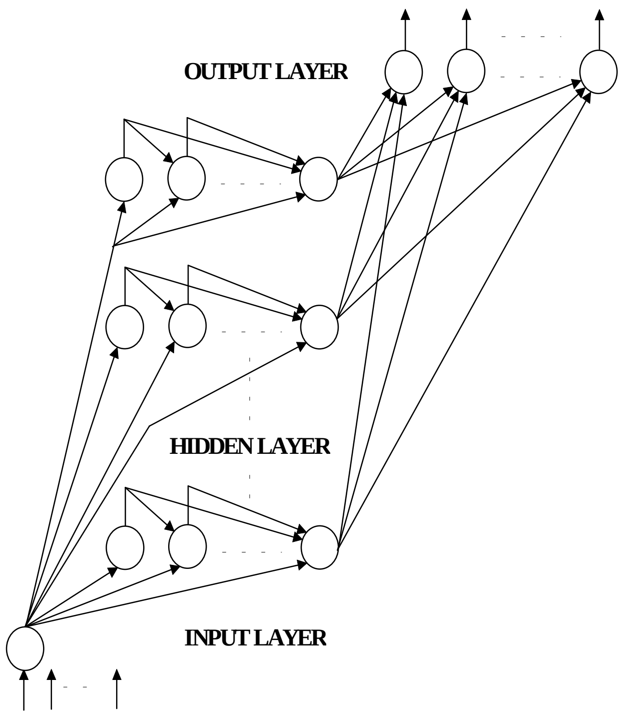

Figure 2. The network architecture.

The unique unit of the *input layer* is fed by the complete sequence of input numbers. Then this layer spreads its output simultaneously to all the units of *the hidden layer*. Once the whole sequence of digits has been processed by the *input* and *hidden layers*, the *output layer* associates the winning group of the *hidden layer* with one concrete telephone number.

2.1 The input layer

The *input layer* carries out the mission of transforming an input number into a normalized vector. As the *STN* [3][4][5] requires its inputs to be normalized, our aim was to incorporate to the architecture a specific and novel layer accomplishing this cue. The *input layer* associates to an input number a vector as shown in figure 3. Any of the so obtained vectors has the property of being unitary (its modulus is equal to one). This is to say, they are already normalized. This distribution of vectors also incorporates the characteristic that the digit *j* is halfway to the digits *j+1* and *j-1*.

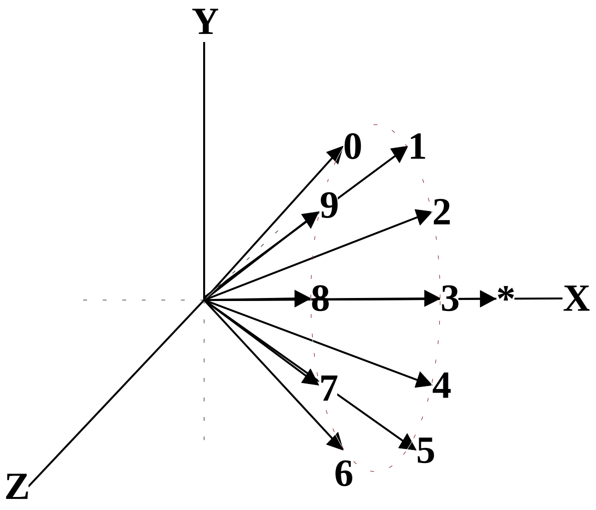

Figure 3. The vectors graphical representation.

The output of the input layer is obtained by the following mathematical expression (2.1.1):

if (j = '')*

then $N(j) = (1, 0, 0)$

else $N(j) = (\cos\alpha,\ \sin(j*\frac{\pi}{5}) * \sin\alpha,\ \cos(j*\frac{\pi}{5}) * \sin\alpha)$

where *j* is the input digit to the layer, and α is the angle formed by the vector and the *X*-axis. By giving larger values to the angle, the *STN* confuses less the vectors that it receives of the *input layer.* This is due to the fact that a larger value of α implies that the vectors are more separated from each other, so the scalar product of the input vector by the learned vector will give a smaller value as a result.

In our application, the *input layer* associates each possible input digit, *0* to *9*, and the generic digit *, to a normalized vector according to expression (2.1.1). The angle $\pi/5$ is obtained by dividing the circumference length

(*2π*) by *10* (the maximum number of possible input digits). Obviously, expression (2.1.1) may be generalized for other applications where the input number could vary from *0* to *n*, just substituting the angle *π/5* by $2\pi/(n+1)$. For the generic digit we have chosen the vector *(1,0,0)* because it is halfway to the rest of the normalized vectors.

2.2 The hidden STN layer

In our architecture this layer corresponds to a *spatio-temporal network*, made up of *n*m* units, where *n* is the number of sequences of digits (or telephone numbers) to be learned, and *m* is the number of elements (or digits) making up a sequence.

Just as in other networks, these network units receive an input value *net* obtained as the scalar product of the input vector by the unit's weight vectors: *Q*Z*. Furthermore, each unit receives an input signal from the outputs of all the preceding units. Preceding means to the left of a given unit, as represented in figure 1. Since all the weights associated to the last connections have the value of a constant *d < 1*, the total input of unit *i* will be:

$$I_i = Q_{1i} * Z_i + d \sum_{k=1}^{i+1} x_k \qquad (2.2.1)$$

where x_k is the output of unit *k*. The output of unit *i* is modeled by means of a differential equation as:

$$\dot{x}_i = A(-a x_i + b\left[I_i - \Gamma\right]^+) \qquad (2.2.2)$$

where *a* and *b* are both positive constants. Function *[u]+* is defined as:

$$[u]^+ = \begin{cases} u & if\ u > 0 \\ 0 & if\ u \le 0 \end{cases} \qquad (2.2.3)$$

where Γ represents a value threshold. We will consider that Γ is a previously defined constant value that filters the total input of the unit, only affecting the values that overcome it. Function *A(u)* is a so called *attack function*. It is defined by equation:

$$A(z) = \begin{cases} z\ if & z > 0 \\ cz\ if & z \le 0 \end{cases}$$

where *0 < c < 1*. The *attack function* is used to determine that the times of charge and discharge of the output values of the unit's weights are different. The calculated unit's attack function value is multiplied by a small value (*δt*) that indicates the speed of updating the network, producing this way the updating value of the output of the unit. The unit's output is updated adding the calculated attack value to the previous output value.

The *hidden layer* is divided into groups of *m* units. Each of them has to learn and recognize one telephone number. The output of the last unit of each group will be the group's output value.

The output unit of each group presenting the largest answer will be the only with a non null result. In other words, we convert to *1* the highest output value of each group, and to *0* the rest. The output of each group may be calculated as:

$$z_i = \begin{cases} 1 & \text{if } \|x_i\| > \|x_j\| \ \forall j \ne i \\ 0 & \text{otherwise} \end{cases}$$

These outputs will be the inputs to the *output layer*.

Let’s consider now the values given to the parameters in this layer. Parameter *δt* has been assigned a value of *1*, leaving this way the updating value unchanged. Parameter *c* representing the discharge value of the output intensity of each unit, has been assigned a value of *0'1*, so that, once a unit has been charged, it will not be quickly discharged. The total input is filtered with a threshold value Γ. When a unit receives the memorized vector as its input, it gives *1* as a result. So, assigning a value of *0'8* to Γ, only similar vectors to the expected one will be allowed to go through. Parameters *a* and *b*, indicating the importance of the previous output and the total input, respectively, have been assigned the values *1'1* and *1/0'3*, in order to adjust the output intensity of each group. Parameter *d* represents the influence of the connections of the preceding units to the unit's total input. We have given it a value of *0'2*. This way we don't lose the charge of a sequence of guessed digits that can take a maximum value of two, and at the same time we don't annul the total input.

2.3 The output layer

Finally, the *output layer* consists of a series of processing elements called *outstars*. The output of an *outstar* is:

$$\dot{y}'_i = -a y'_i + c\, w_i^{eq}$$

where w_i^{eq} is the fixed value of the weight that has been found during the learning step [3] [1], as it will be described later on.

The *outstar* quickly reaches a balance value equal to the value of the weight of the connection coming from the winning unit of among the outputs of the groups. An easy way of understanding this process consists in realizing that the balance output of the outstar equals the input value *net* of the *outstar*,

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

where z_j is the input received from the corresponding group of the *STN* layer. Since $z_j = 0$, unless $j = i$, it is possible to rewrite:

$$y'^{eq}_k = w_{ki} z_i = w_{ki} \qquad (2.3.1)$$

This simple algorithm uses balance values of the activities and outputs of the nodes. This way we avoid solving numerically all the corresponding differential equations.

2.4 Learning

In this section we will describe the learning algorithms of the different layers of our architecture.

The *input layer* doesn't need a learning phase. The translation is easily performed by implementing the function described in section 2.1.

Learning of each unit of the *STN layer* starts with the initial weight vector *w*, which evolves according to the differential equation $\dot{w} = -cw + dIy$, where y is the output, and *c, d > 0*. The role of an *STN* unit is to memorize a normalized input vector, providing the larger an output intensity the more the input vector resembles the learned input vector.

On the other hand, the weights of each outstar unit of the output layer evolve during their learning process according to the differential equation

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

where $a, b, c > 0$ and $net_i = I * W$.

3. Implementation

Our architecture has been proved as a particular application in the problem of the correction of phone numbers as described in the introduction section. We have implemented a simulator using *the Visual C++ 5.0* programming language. Initially the user is shown a dialogue window where he is invited to dial the desired number. Apart from the normal digits (*0* to *9*), the user has the possibility to dial an asterisk ('*'), if he doesn't remember the corresponding digit. Once the number has been dialed, our architecture processes the input number and responds with an output number. The system has memorized in our latest tests one hundred 9-digit telephone numbers.

Apart from the automatically generated classes by *Visual C++ 5.0,* we have written the classes corresponding to the different layers as well as to the complete network.

The implementation of the *input layer* is very simple. It has been as easy as implementing function (2.1.1) as a method of the *CInput* class.

The *STN layer* is formed by a group of the same type of units. To implement one of theses units, class *CUnit* has been written. This class is formed by a group of attributes and methods that carry out the functions described in section 2.2. The class has been parameterized in order to use dynamic memory. That’s why it incorporates an integer parameter indicating the number of inputs coming from the preceding units of its group. Class *CNetworkSTN* (*spatio-temporal network*) is formed by a list of *CUnit* objects. This class forces each unit to learn and propagate, and finally, the competition among the outputs of the last unit of each group is realized. This competition is nothing but the change to one of the output of the unit with the largest intensity, and to zero of the others.

The *outstar output layer* consists of a group of objects of class *COutstar*. This class uses formula (2.3.1) to calculate its output. With regard to learning, to each $outstar_i$ of the *output layer*, our class assigns to each weight w_i the corresponding value to each input vector y_{ij}.

The complete architecture uses the already described classes *CInput*, *CNetworkSTN* and *COutstar*, forming a general class *COurArchitecture*, with a parameterized constructor with two parameters that indicate the number of groups and the number of elements of the input sequence, respectively. That is to say, the class *COurArchitecture* may contain a variable number of groups to learn as well as of elements of these groups. This way, this design allows us to implement flexible architectures.

4. Results

For the verification of results we have carried out several simulations with 100 different telephone numbers composed of 9 digits each. With the purpose of obtaining a complete statistic of results we have introduced all the telephone numbers with some error, and with different values for α. The studied cases are the following ones: the telephone number is correct (**C**), a generic digit appears in a random position (**1***), two generic digits are introduced in random positions (**2***), three generic digits are dialed in random positions (**3***), one erroneous digit is dialed in a random position (**E1**), there occur two errors in random positions (**E2**), three errors appear in random positions (**E3**), two digits are exchanged in a random way (**I2**), the telephone number is displaced one position to the right from the initial position (**D1**), and the phone number is displaced one position to the right from a random position (**DX**). The simulations present as a result the three telephone numbers that took the greatest output intensities.

Table 1 shows the number of successful guesses in the simulations, with the following values of parameter α in the input layer: *π/4*, *π/3*, *2π/5* and *π/2*.

α		π/4	π/3	2π/5	π/2
C	**1st**	100	100	100	100
	2nd	0	0	0	0
	3rd	0	0	0	0
1*	**1st**	87	86	90	91
	2nd	7	12	5	6
	3rd	3	1	4	3
2*	**1st**	64	65	72	69
	2nd	15	12	14	13
	3rd	6	14	4	6
3*	**1st**	45	44	54	48
	2nd	14	21	20	17
	3rd	12	9	7	13
E1	**1st**	78	84	90	88
	2nd	14	12	5	8
	3rd	3	3	2	2
E2	**1st**	57	59	62	66
	2nd	15	21	13	11
	3rd	6	7	9	6
E3	**1st**	39	36	39	40
	2nd	12	17	14	17
	3rd	11	8	9	4
I2	**1st**	65	74	72	72
	2nd	13	9	7	14
	3rd	4	2	9	8
D1	**1st**	83	83	84	83
	2nd	8	9	7	8
	3rd	3	3	4	3
DX	**1st**	70	78	79	72
	2nd	10	10	7	14
	3rd	8	4	6	4

Table 1. Simulation results.

where:

- **C:** the system's number of successful guesses introducing the correct telephone numbers.
- **1*:** the system's number of successful guesses introducing the telephone numbers with a generic digit.
- **2*:** the system's number of successful guesses introducing the telephone numbers with two generic digits.
- **3*:** the system's number of successful guesses introducing the telephone numbers with three generic digits.
- **E1:** the system's number of successful guesses introducing the telephone numbers with one random erroneous digit.
- **E2:** the system's number of successful guesses introducing the telephone numbers with two random erroneous digits.
- **E3:** the system's number of successful guesses introducing the telephone numbers with three random erroneous digits.
- **I2:** the system's number of successful guesses introducing the telephone numbers with two exchanged digits.
- **D1:** the system's number of successful guesses introducing the telephone numbers displaced one position to the right from the initial position.
- **DX:** the system's number of successful guesses introducing the telephone numbers displaced one position to the right from a random position.

If we observe the results presented on table 1, we can confirm that, in most cases, for larger values of the angle α the obtained results are better. This is because the system confuses less the digits as their corresponding vectors are more distant. On the other hand, it is possible to appreciate that the system's behavior is better for telephone numbers containing asterisks ('*') than for telephone numbers with erroneous digits. Consider also that the number of successful guesses in case **D1** is almost of 85% in the telephone number with a greater output intensity. Finally, for any value of α, and in most test cases the system's number of successful guesses is equal or greater to a 70%. It is necessary to keep in mind that in cases **E3** and specially **3***, the telephone numbers obtained as an answer cannot be considered to be erroneous, as you may have several telephone numbers that fulfill the conditions of the dialed telephone number.

On the other hand, it is interesting to analyze the complexity of the implementation of our architecture. The complexity is a function of two parameters: *n*, that is the number of telephone numbers of the agenda, and *m*, that expresses the number of digits of each telephone number. We will analyze the complexity of our implementation layer by layer.

The *input layer* is just the implementation of the mathematical function (2.1.1) of order *O(1).* As it is passed *m* digits, its complexity will be of order *O(m).* The *STN layer* is composed of *m*n* units, and to each of these units the *m* digits are passed; so, the complexity is of order $O(m*n*m) = O(m^2*n)$. The *outstar* is governed by the equation $y_k^{'eq} = \sum_j w_{kj} z_j$; as this sum goes from *1* to *n*, the order of complexity would be *O(n).* Finally, summing all three layers, the complexity of the system would be $O(m)+ O(m^2*n)+O(n)$. As *O(m)* and *O(n)* are insignificant towards $O(m^2*n)$, the total complexity will be of order $O(m^2*n)$. The extra cost to this complexity is the cost of the learning step which is only carried out in one occasion. The complexity of learning in the *STN layer* is of order $O(m*n)$, since each unit of the *STN layer* memorizes a normalized vector of three components, and this is of order $O(1)$. On the other hand, the *outstar* memorizes *n* digits (assignment of *n* values in an array of *n* components), and cause we have *m outstar* units, the complexity will be of order $O(m*n)$. So the complexity of learning is $O(m*n)$. If we compare this complexity with a common method, -just keep in mind the case of the displacements- the complexity would be very superior to ours, hardly inferior to $O(n^2)$.

Furthermore, our implementation is able to evaluate many more cases than those presented in this paper, but always remaining the same complexity, while in any other algorithm you have to evaluate case by case.

5. Conclusions

An architecture that includes, as a relevant novelty, an *input layer* associating to an input number a normalized vector by means of a mathematical function, has been presented. This vector hardly “confuses” with any other representative vector of any other input number. This way the *STN layer* takes a great advantage in recognizing the number sequences.

With regard to the results of the simulations, consider that, for any value of α, and in almost all the cases of our tests, the percentage of successive guesses of the system takes a value of 70% or higher. In particular, the percentage in tests with telephone numbers with displaced sequences of digits, takes a surprising value around an 80% (if we only take in account the winner telephone number).

The complexity of the implementation of the proposed architecture is lower compared to any common method. Especially if we consider the displacement cases. Our implementation is valid in much more cases than those analyzed in the tests, always presenting the same complexity.

Finally, we want to highlight that, thanks to the flexibility of our architecture, it is applicable to a great variety of problems related with the recognition of number sequences, like in the case of artificial vision, speech recognition, and so on.

References

[1] S. Grossberg. Studies of Mind and Brain, volume 70 of Boston Studies in the Philosophy of Science. D.Reidel Publishing Company, Boston, 1982.

[2] S. Grossberg. Learning by Neural Networks. Published by Stephen Grossberg in Studies of Mind and Brain. D. Reidel Publishing, Boston, MA, 65-156, 1982.

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

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

[5] R. Hecht-Nielsen. Nearest matched filter classification of spatio-temporal patterns. Technical Report, Hecht-Nielsen Neurocomputer Corporation, San Diego, CA, 1986.