

Neural Crystal A Dynamic Recurrent Network

Frédéric D. Jordan

Commissariat à l'énergie atomique DMA/AIM
Centre de Limeil-Valenton, FRANCE

1. DESCRIPTION

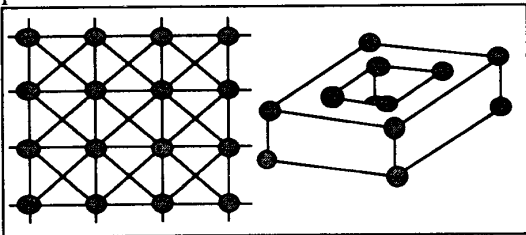
1.1. Introduction

Unlike feedforward MLP, recurrent networks have ability to work with temporal inputs and outputs. So the signal can be input directly to the network without pre-processing like time-windowing and without having direct dependency between signal duration and number of neurons.

Crystal organisation of neurons has already been proposed by a number of neurobiologists [1] [2] but less work has been done to study properties of such ANNs. An important difference with recurrent multilayered networks lies in the 8 isotropic directions of the structure: it has the same topological organisation along 8 directions. The motivation for the architecture described in this paper is then to propose a model which presents the following properties: Hardware feasibility, compact matricial formalism and temporal dynamic. So we present a mathematical formalism and four examples of application in robotic and signal recognition.

1.2. Architecture

Neurons are connected within a d-dimensional cubic structure, so each neuron is connected to its $(3^d - 1)$ neighbours. This connection pattern is repeated for all the neurons to create a d-dimensional hypercube. The number n of neurons on cube side defines a n^d sized network. The side neurons are edge to edge connected so that the structure can be visualised as a d+1 dimensional torus. Following figure shows a 2d, 4x4 sized network and its toroidal representation:



1.3. Transfer Function

Each neuron computes a weighted summation of its inputs, a low frequency filtering and a non-linear activation function (sigmoid). The neuron has a feed-back with a w_{00} weight. So the Laplace transform of the filter is written as:

$$g(p) = \frac{1}{1 + \tau p} \text{ so in closed loop by } w_{00} \text{ with low signals}$$

$$s(p) = \frac{e^{(p)/(1-w_{00})}}{\left(1 + \tau p / (1-w_{00})\right)}$$

The neurone stability can then be adjusted using w_{00}

$$\text{to obtain a time constant } \tau' = \frac{\tau}{1 - w_{00}}$$

1.4. Relaxation Equation

The neuron states at time t are represented by the $s_{ij}(t)$ elements of a $n \times n$ matrix $S(t)$. Neuron inputs are represented by the $i_{ij}(t)$ elements of a $n \times n$ matrix $I(t)$.

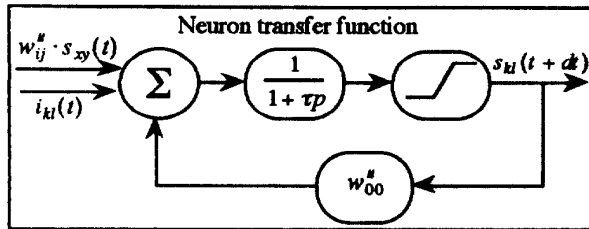
Weights are the elements of a $n \times n$ matrix which is composed of 3×3 matrices. Each 3×3 matrix W_{ij} represents the weights of input connections to the ij -neuron. In the particular case of a 4x4 2-dimensional network, we get:

(with $\bar{I} = -1$)

$$W = \begin{pmatrix} W_{00} & W_{01} & W_{02} & W_{03} \\ W_{10} & W_{11} & W_{12} & W_{13} \\ W_{20} & W_{21} & W_{22} & W_{23} \\ W_{30} & W_{31} & W_{32} & W_{33} \end{pmatrix} \text{ with } W_{kl} = \begin{pmatrix} w_{11}^k & w_{10}^k & w_{11}^k \\ w_{01}^k & w_{00}^k & w_{01}^k \\ w_{11}^k & w_{10}^k & w_{11}^k \end{pmatrix}$$

$$\text{and } S(t) = \begin{pmatrix} s_{00}(t) & s_{01}(t) & s_{02}(t) & s_{03}(t) \\ s_{10}(t) & s_{11}(t) & s_{12}(t) & s_{13}(t) \\ s_{20}(t) & s_{21}(t) & s_{22}(t) & s_{23}(t) \\ s_{30}(t) & s_{31}(t) & s_{32}(t) & s_{33}(t) \end{pmatrix}$$

With this structure there is no zero element in weight matrices. This compactness property offers a good use of the memory allocated for matrices storage.



For 2-dimensional networks, relaxation operation can then be written with a matrix product *:

$$S(t + dt) = f \circ g(W * S(t) + I(t)) \text{ with } * \text{ defined by:}$$

$$s_{kl}(t + dt) = f \circ g \left(\sum_{i=-1}^1 \sum_{j=-1}^1 w_{ij}^{\#} \cdot s_{xy}(t) + i_{kl}(t) \right)$$

with $x = (k + i) \text{ modulo } n$ and $y = (l + j) \text{ modulo } n$.

Several symmetries of the weight space can be expressed using this formalism [4]. Notice that with initial states equal to zero (as we used in the following) we have:

$$f \circ g(-I(t)) = -f \circ g(I(t))$$

2. LEARNING ALGORITHMS

2.1. Introduction

Weights are randomly initialised between [-0.1;1]

We note $(S_1, S_2, \dots, S_p)(t)$, the p outputs computed at time t . If we know the expected output $(Y_1, Y_2, \dots, Y_p)(t)$, we can define error as a distance $d(S_k, Y_k)$ (for all k). In general, we choose:

$$E_Y = \sum_p \int_0^{T_{max}} e^{|S_p(t) - Y_p(t)|} \cdot dt$$

If we don't know Y_k , a function h must be found such as we have: $E = d(h(S_k), h(Y_k))$.

Several algorithms have been tested [5] in order to minimise E , but we focused on 2 simple minimization techniques which are non-local and supervised.

2.2. Sequential implementation

Let h , h_{init} and h_{min} 3 positive values, with $h_{init} > h_{min}$.

For each matrix element w , we do:

0. $h = h_{init}$.
1. Compute $E(w+h)$ and $E(w-h)$.
2. Keep w in order to have the Min of $(E(w-h), E(w), E(w+h))$.
3. $h = h/2$.
4. Go back to 1. until $h < h_{min}$.

Typical values we used are $h_{init} = 1$ and $h_{min} = 0.1$

Notice that learning time increases at least linearly with the number of neuron since weights are updated individually.

2.3. Parallel implementation

This algorithm was developed for training a 8x8 network on a massively parallel supercomputer Cray T3d. It includes 128 Alpha processors with 64Mo per processor. We used 64 processing elements. For relaxation, each has the task to calculate a neuron state in respect to its neighbours. At $t = t_{max}$ we get the error for this particular set of weights. The following learning algorithm was implemented in each processing element (k,l) :

0. $h = h_{init}$.
1. Choose a random 3x3 matrix R_{kl} in range $[-h;h]$.
2. Compute $W_{kl} = W_{kl} + R_{kl}$ for each (k,l) .
3. Compute relaxation.
3. If $error < error_{min}$, go back into 1.
4. Else, $h = h/2$ and $W_{kl} = W_{kl} - R_{kl}$
5. Go back in 1. until $h < h_{min}$.

3. SIGNAL GENERATION

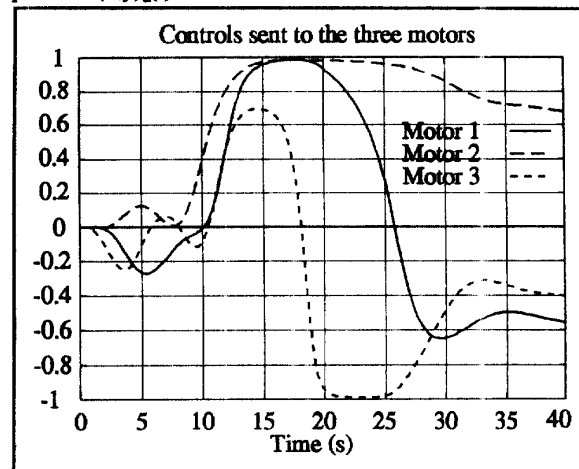
3.1. Introduction

The ability of the network to produce complex transient responses with simple input signal can be used to synthesize any temporal command. In the following, we show two applications where the network has to control the robot arm simulated motion.

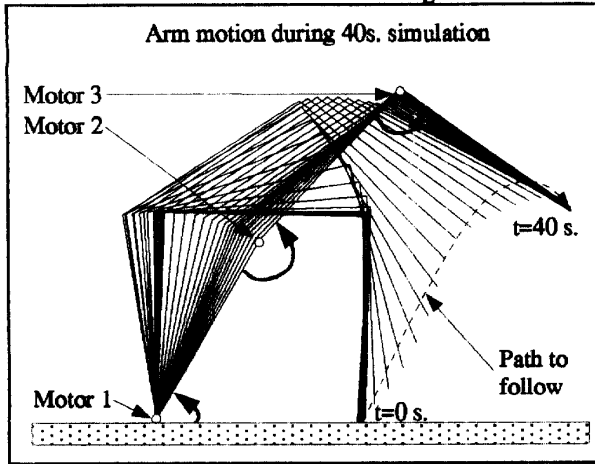
3.2. Following a given path

The generation of robot trajectories has been studied with various solutions [8] [9]. The following application shows that it is possible to directly use the Neural Crystal dynamic to perform such a task.

A 4x4 network is trained to generate the 3 motors control of a robot arm in order to follow a given path. So, 3 outputs of the network provide the angular rotation speed of motors; with these data we compute what would be the arm position $(x, y)_c(t)$ and we compare it to the expected position $(x, y)_d(t)$ to define error.

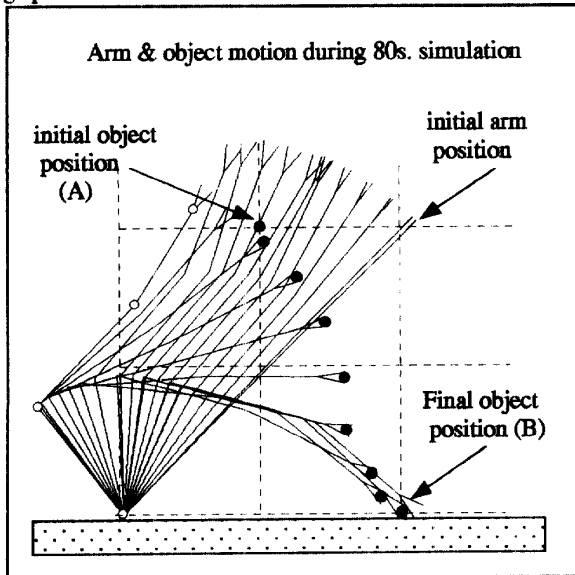


The neural crystal is activated with a 5s duration pulse. Figures show motion of the arm and the corresponding controls sent to the 3 motors after learning.



3.3. Making a given task

In this example, we trained a network to succeed in the following task: to take an object in point A and drop it in point B in less than 80s. of simulation. We used a 8x8 network, with input in neuron (0,0) and 4 outputs in (3,4)-(4,3)-(4,5)-(5,4). The fourth output is the command of the grip.



We note $\alpha_p(t)$, the opening angle of the grip in radian.

T is the position reached by the arm tip when $\alpha_p(t)$ decreased for the first time. The object is then considered as taken. G is the position where $\alpha_p(t) > 0.5 \pi$. after the object has been taken. The object is then considered as dropped. We define the error as: $E=AT^2+BG^2$

So, $E=0$ implies that the robot takes the object where it is in A(1,2) and drops it where we want in B(2,1).

Initial angles are: $\alpha_1=\pi/4$; $\alpha_2=\alpha_3=\pi$; $\alpha_p=0$. Activation is done with a positive 2s. duration pulse. We used the parallelized learning algorithm for training. It took about 100 000 relaxations to get the following result (At the rate of 200 relax/s).

4. SIGNAL RECOGNITION

4.1. Introduction

When used for recognition of signals, the network has to learn how to produce an output $Y^P(t)$ for a set of input signals $I^P(t)$. Of course, $Y^P(t)$ must be easier to recognize than $I^P(t)$. In the following we will use square output signals.

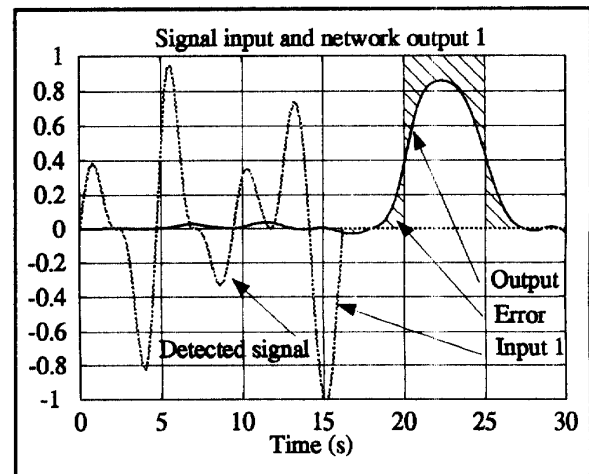
4.2. Elementary recognition

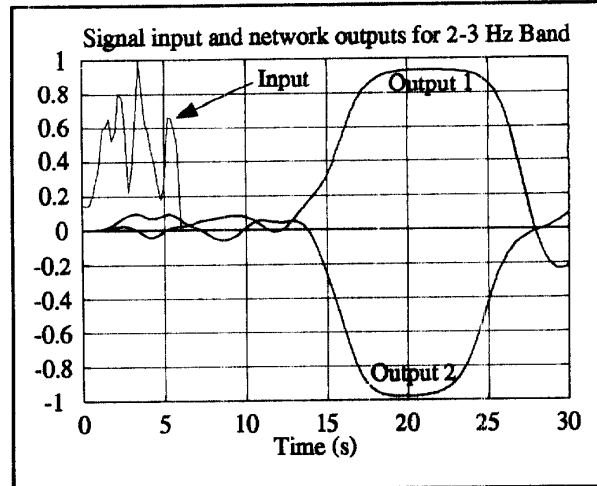
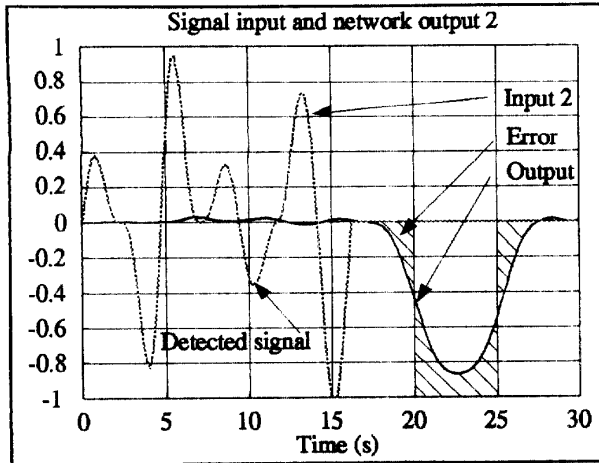
The use of this network for recognition tasks is based on its capacity to produce a typical transient response.[7]

So the question may be asked: Is the network able to recognize a signal inside another or is it only sensitive to the beginning or to the end of the input signal?

This would respectively means that the network is intrinsically too unstable or too stable to achieve any complex signal recognition.

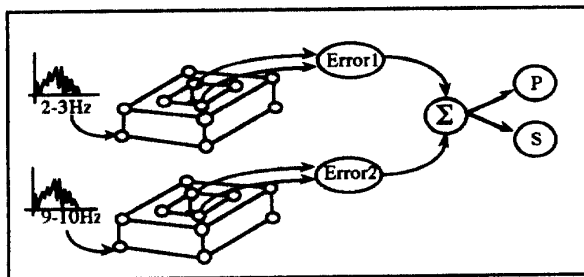
Following figure shows that a trained network succeed to produce a positive or negative squared output signal for





4.3. Seismic signal recognition

We studied the classification of seismic signals in 2 sets S and P which are related to 2 peculiar propagation modes of the seismic waves. Two Neural Crystals are used: one takes the 2-3Hz band-pass seismogram as input and the other the 9-10Hz Band. To produce the "output error", we add errors made by each crystal.



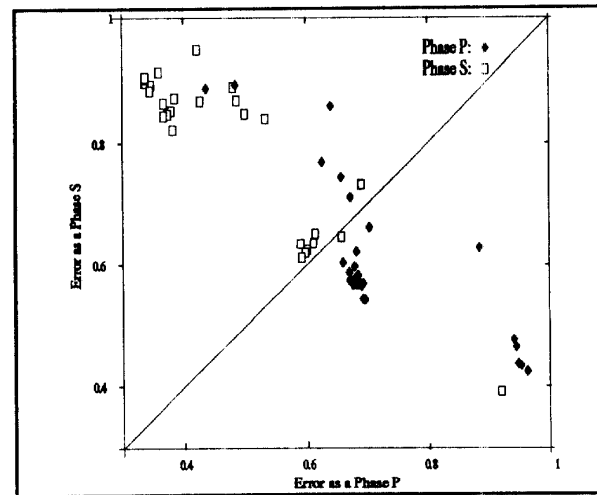
For a given input we calculate then:

The output error E_s which evaluates the difference with a typical output S and the output error E_p with a typical output P. Then, if $E_p > E_s$, we conclude that the input signal was a S phase (and resp. for P phase).

The following picture shows E_p versus E_s for each seismic signal. The $y=x$ line separates the 2 types S and P.

Learning has been made on 12 P-samples and 12 S-samples recorded from real seismic events on a station A. We trained the network using the previously described algorithm with couples of P and S phases of the same event. 83% of these signals were correctly sorted after learning.

In order to evaluate generalisation capabilities, 56 samples of a station B were given to the network. Results are 77% of correct sorting which is rather good, taking account of particular distortions which are station dependent.



5. CONCLUSION

We have created and tested a simple and compact neural structure which can complete sorting task on time dependent signals. An application for real seismic signals classification has been developed. We showed that it was also able to produce temporal command for robotic applications.

Because implementation of relaxation and learning algorithm has been possible using parallel programming, we got high level computing performance using the Massively Parallel Processor Cray T-3D.

We would like to thank Mr. G. Clément for useful discussion on learning algorithm and Ms. C. Leuhan for C language software development.

6. REFERENCES

- [1] J.P. Changeux, L'homme neuronal, *FAYARD*, 1983.
- [2] Wässle, H., Peichl, L. & Boycott, B., Dendritic territories of cat retinal ganglion cells, *Nature* 292, 1981.
- [3] M. M Snyder and D. K. Ferry, Open-Loop Stability for Layered and Fully-connected NN, *Proc. of nEuro'88*.
- [4] F. D. Jordan and G. Clément. Using the Symmetries of the MLP Structure to reduce the Weight Space, *Proc. of IJCNN, Seattle*, 1991.
- [5] R. Williams and D. Zipser, A Learning Algorithm for Continually Running Fully Recurrent Neural Networks, *Neural Computation*, 1988.
- [6] Davalo and Naïm, Des réseaux de neurones, *EYROLLES*, 1993.
- [7] Kumpati S. Narendra, Identification and Control of Dynamical Systems, *IEEE Transactions on N.Net.* 90.
- [8] G. Josin, D. Charney and D. White, A Neural-Representation of an Unknown Inverse Kinematic Transformation, *Proc. of nEuro'88*, 1988.
- [9] Davidor, Y. , Robot programming with a genetic algorithm, *IEEE International Conference Proc. on Computer Systems and Software Engineering*, 1990.