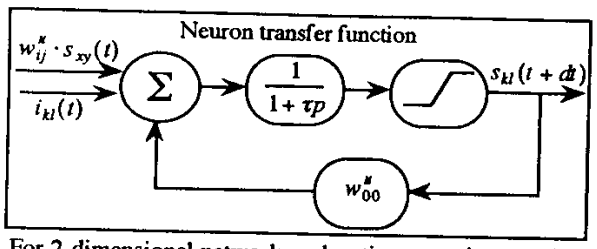


For 2-dimensional networks, relaxation operation can then be written with a matrix product *:

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

$$s_{kl}(t+dt) = fog\left(\sum_{i=-1}^{1}\sum_{j=-1}^{1} w^n_{ij} \cdot s_{xy}(t) + i_{kl}(t)\right)$$

with $x = (k+i)$ modulo n and $y = (l+j)$ 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:

$$fog(-I(t)) = -fog(I(t))$$

2. LEARNING ALGORITHMS

2.1. Introduction

Weights are randomly initialised between [-0.1;1]

We note (S1,S2,...,Sp)(t), the p outputs computed at time t. If we know the expected output (Y1,Y2,...,Yp)(t), we can define error as a distance d(Sk,Yk) (for all k). In general, we choose:

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

If we don't know Yk, a function h must be find such as we have: E=d(h(Sk),h(Yk)).

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, hinit and hmin 3 positive values, with hinit>hmin.

For each matrix element w, we do:

0. h=hinit.
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<hmin.

Typical values we used are hinit=1 and hmin=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 Apha 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=tmax we get the error for this particular set of weights. The following learning algorithm was implemented in each processing element (k,l):

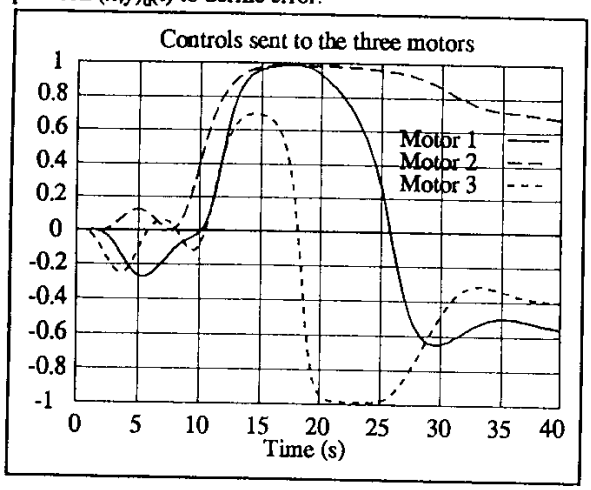
0. h=hinit.
1. Choose a ramdom 3x3 matrix R_{kl} in range [-h;h].
2. Compute $W_{kl}=W_{kl}+R_{kl}$ for each (k,l).
3. Compute relaxation.
3. If error<erormin, go back into 1.
4. Else, h=h/2 and $W_{kl}=W_{kl}-R_{kl}$
5. Go back in 1. until h<hmin.

3. SIGNAL GENERATION

3.1. Introduction

The ability of the network to produce complex transcient 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.