Physical Models that Learn

Nicolas Szilas, Claude Cadoz

Physical Models That Learn

Nicolas SZILAS
nicolas.szilas@imag.fr

Claude CADOZ
claude.cadoz@imag.fr

ACROE-LIFIA
INPG - 46, av. Felix Viallet
38031 GRENOBLE Cedex - FRANCE

Abstract

In this paper is proposed an analysis method for Physical Model networks, based on connectionist learning algorithms. The first experimental results, obtained on limited cases, are quite encouraging and suggest some improvements, using the interactivity of Physical Models.

1 From synthesis to analysis

Synthesis and analysis form a pair of two symmetrical processes: while *synthesis* turns computer parameters into sound, *analysis* attemps to infer, from a sound behaviour, the parameters of the model in the machine. Given a physically based sound *synthesis* system, this paper deals with *analysis* techniques that have to be associated with it.

CORDIS [Cadoz *et al.*, 1993] is a physical description language for music instruments. It has specificities that generate some constraints in the corresponding analysis system:

- instead of considering large differential equations from the instrument to be modeled and then numerically integrating them, it manipulates a large number of *simple* physical automata, which represent small "pieces of matter", with which it "reconstructs" the object. Thus, the music instrument is represented by a network of punctual masses linked together by elastic, viscous and nonlinear elements of various types (fig. 1). Therefore, the purpose is to find analysis methods which can be implemented in a network similar to the one used in synthesis. Traditional identification techniques do not provide such a method.

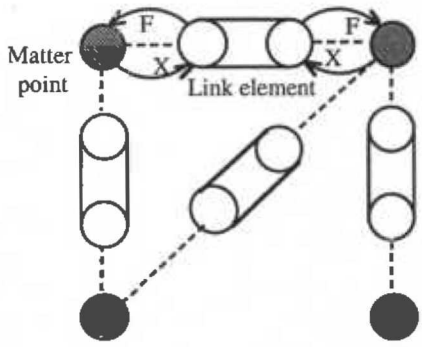

Figure 1: a CORDIS network

- In order to keep up a central property of physical objects, any CORDIS unit is **bidirectional**: it does not receive any information (forces for example) without sending other information (displacements for example). Thus, the analysis process is not conceived as "information processing" but as an "interacting process", in bidirectional communication with its environment. Furthermore, this environment not only contains the instrument to be analysed but also the instrumentalist himself, since sound behaviour depends both on the instrumentalist action and the instrument response. This leads us to the following interaction diagram (fig. 2):

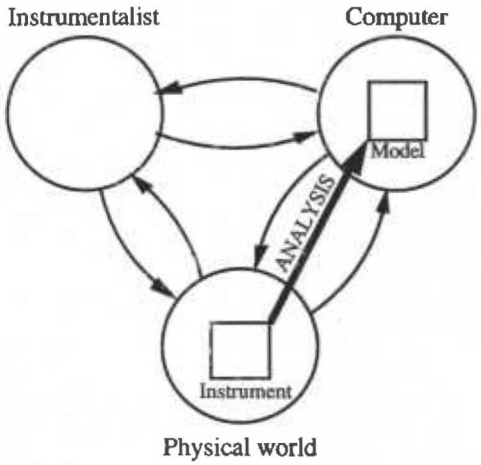

Fig. 2: the interacting process of analysis

- CORDIS enables real-time simulation of physical models ... so has to do the analysis...

Three main characteristics of the *physical models* of CORDIS have just been described: the use of many simple processing units, the bidirectionality and the real-time capabilities; It turns out that these tend also to be significant features of many *cognitive models* (Artificial Intelligence becomes more and more Distributed Artificial Intelligence, Multi-agent or Neural Networks; interactivity and real-time are main features of Human-Computer Communication, Knowledge Acquisition and Autonomous Robotics). In particular, Neural Networks offer learning algorithms dedicated to the modification of the parameters of a network according to its environment.

The "physical models that learn" decribed here are mechanical networks endowed with the learning abilities of neural networks. In so far as they interact with the instrumentalist and the instrument, they should be able to reproduce the mechanical behaviour of the instrument.

2 Neural networks

The Neural Networks group together several kinds of algorithms, implemented in structures inspired by neurobiological data. A connectionist (neural) network consists of many simple units connected with each other. All the acquired knowledge of the network lies in these connections. To each of them is assigned a *weight*, qualifying the influence of a unit upon the other (fig. 3).

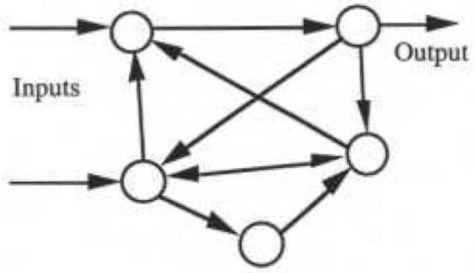

Figure 3a: a neural network

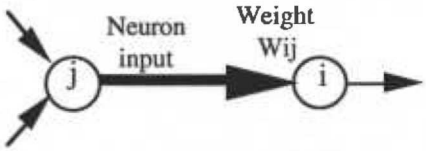

Figure 3b: a connection

A network has an activation rule (specifying the way the outputs are calculated) and a learning rule (specifying the way the weights are updated), which are equivalent to synthesis and analysis.

The comparison between the CORDIS networks and the connectionist networks leads to the following remarks:

- The two types of networks are very similar. Thus, the mapping between them will be simple: a neural unit corresponds to a punctual mass or a visco-elastic element (fig. 4).

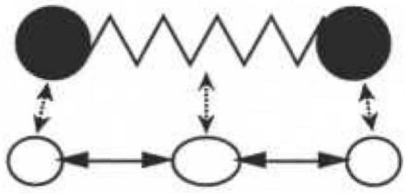

Figure 4: mapping between neural and CORDIS networks

- In the large spectrum of connectionist researches, the learning CORDIS networks are concerned only with a specific class of neural networks: the *supervised recurrent neural networks with hidden units* (that is networks which have units that are neither inputs nor outputs, whose connectivity contains cycles, and which learn how to reproduce an external signal or behaviour). These networks process temporal sequences. In this paper, they will be called *dynamic networks*.

Here is a brief historic of dynamic networks... In 1985 was proposed the very famous Back-Propagation Algorithm for feedforward networks. Since any recurrent architecture is equivalent to a feedforward architecture (in a finite time), this algorithm is easily adaptable for dynamic networks (it is then called Back-Propagation Through Time, ie BPTT [William and Peng, 1990]). Unfortunately, this algorithm needs to store all the temporal historic of the network, which is quite expensive. Later, several algorithms were proposed to cope with this problem but they only concerned limited cases (learning of a fixed point or special architectures). Eventually was proposed an algorithm that needs no storage , which is therefore particularly well suited for real-time applications (it is called Real-Time Recurrent Learning, ie RTRL [William and Zipser, 1989]). Since 1989, several improvements have been studied, but nothing quite new has been found. The major drawbacks of the RTRL algorithm are its nonlocality and its large amount of storage (spatial storage). Nevertheless, this is the one used in this paper because of its real-time capabilities.

3 The Algorithms

Although Neural Networks are able to deal with nonlinear processing units, the present study is limited to linear vibrating structures. Furthermore, only non-dissipative structures are studied here.

The synthesis formulas are established by discretising the laws of mechanics:

- a punctual mass is an automaton receiving forces and yielding one position, according to the fundamental principle of dynamics:

$$X_i(n) = 2X_i(n-1) - X_i(n-2) + \frac{1}{m_i}\sum_i F_{ji}(n) \quad (1)$$

where $X_i(n)$ is the position of the mass i at time n, $F_{ji}(n)$ the force applied to the mass i by the spring located between the masses i and j, and m_i the value of the mass i;

- the link element is an automaton receiving two positions and yielding two opposite forces, obeying the law of springs:

$$F_{ij}(n) = k_{ij}(X_i(n) - X_j(n)) = -F_{ji}(n) \quad (2)$$

where k_{ij} is the stiffness of the spring between the masses i and j.

Suppose that the desired behaviour $Xd_i(n)$ of one of the masses in the network, be known. This mass is called the access point.

Then define the error E_{tot} between the actual signal and the desired signal by:

$$E_{tot} = \sum_{n=1}^{N} E(n) \ , \ E(n) = \left[X_i(n) - Xd_i(n)\right]^2 \quad (3)$$

where N is the total length of the simulation.

The purpose of the learning algorithm is to adapt the parameters (masses or stiffnesses) in order to minimise E_{tot}. First has to be studied the minimisation of E(n). As for the classical Back-Propagation algorithm, this minimisation is achieved by a steepest gradient descent: the error is a function of the parameters, which can be represented by a *surface* in a multidimensional space (fig. 5); the current error (with the current parameters) is a point on this surface; the principle of the algorithm is to move this point to a local minimum of the surface, by following the direction in which the slope is the steepest, that is the *gradient* of the error; consequently, the parameters are modified according to:

$$\Delta\vec{p} = \overrightarrow{p(n)} - \overrightarrow{p(n-1)} = -\alpha\overrightarrow{\nabla E(n)} \qquad (4)$$

where α is the learning rate (when α is too small, the convergence is slow, but when α is too great, one can skip over the minimum).

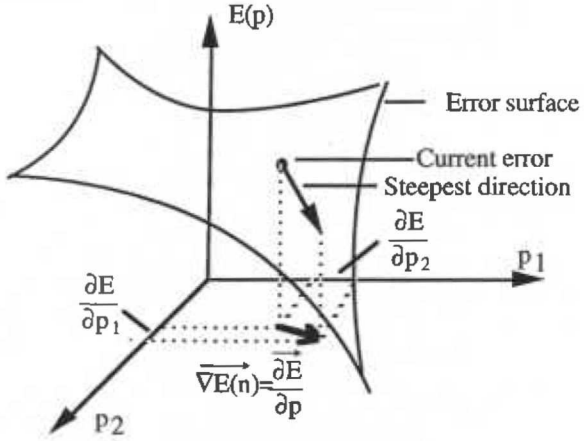

Figure 5: gradient descent

In the case of CORDIS networks, this leads to the definition of four signals:

$$S_i^j(n) = \frac{\partial X_j(n)}{\partial(1/m_i)} \qquad R_i^{kj}(n) = \frac{\partial F_{kj}(n)}{\partial(1/m_i)}$$

$$\sigma_{ij}^l(n) = \frac{\partial X_l(n)}{\partial k_{ij}} \qquad \rho_{ij}^{kl}(n) = \frac{\partial F_{kl}(n)}{\partial k_{ij}}$$

By differentiating the equations (1) and (2) with respect to the parameters, one obtains:

$$* \quad S_i^j(n+1) = 2S_i^j(n) - S_i^j(n-1) + \frac{1}{m_j}\sum_l R_i^{lj}(n) + \delta_{ij}\sum_l F_{li}(n) \qquad (5)$$

$$* \quad R_i^{lj}(n) = k_{lj}(S_i^l(n) - S_i^j(n)) \qquad (6)$$

$$* \quad \sigma_{ij}^l(n+1) = 2\sigma_{ij}^l(n) - \sigma_{ij}^l(n-1) + \frac{1}{m_l}\sum_h \rho_{ij}^{hl}(n)$$

$$* \quad \rho_{ij}^{hl}(n) = k_{hl}(\sigma_{ij}^h(n) - \sigma_{ij}^l(n)) + \delta_{(ij),(hl)}(X_i(n) - X_j(n))$$

where δ is the kronecker symbol.

By applying (4) to (3), one obtains the learning rules:

$$\Delta^1\!/_{m_i}(n) = -2\alpha(X_l(n) - Xd_l(n))S_i^l(n)$$

$$\Delta k_{ij}(n) = -2\alpha(X_l(n) - Xd_l(n))\sigma_{ij}^l(n)$$

where "l" is the access point.

These formulas specify how to move the parameters m and k at each time step n.

It is quite interesting to note that (5) and (6) form a pair of equations very similar to (1) and (2). Thus, $S_i^j(n)$ and $S_i^{lj}(n)$ can be considered respectively as positions and forces of a mechanical network similar to the first one (fig. 6) associated with the mass i and excited by a force: $m_i \sum_l F_{li}(n)$.

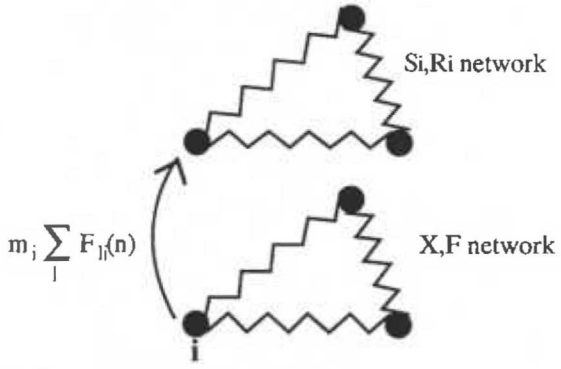

Figure 6: structure of a physical model that learns

To succeed in learning, the initial network is duplicated many times. Therefore, due to the parallelism of the implementation, the analysis can be carried out as fast as the synthesis. However, it requires structures that are more complex than the resulting structure.

Note that this algorithm is not interactive, since the network does not influence the environment.

4 Testing and results

Instead of being implemented directly on a real-time analysis system, the algorithms are tested in more restricted situations:

- there is no actual speed constraint;
- the analysed object, instead of a real instrument, is a computer simulated object, that is a CORDIS object (it is still called the real object);
- the *structures* of the learning object and the real object are the same (only the parameters differ). Thus, it is guaranteed that a perfect solution exists;

- the structures are simple: from one to four parameters to be adapted (fig. 7);

Figure 7a: the basic cell

Figure 7b: the line

- the simulations are very short: up to 50 samples.

Among the many possible experiments, three types of results are set out in this section:

- An empirical study of the local convergence: given an initial learning line, it is essential to test if it is able to find the parameters of a close real line whatever its relative position in the parameters space. In this test, four masses are adaptative. Each of these can be greater or smaller than the corresponding mass in the real object: this leads to 16 tests. In all cases, the learning is successful, which empirically demonstrates the local convergence.

- The limit of the convergence: in the case the learning object and the real object are too far from each other, the learning fails. In a series of experiments with basics cells, the initial stiffness of the learning object was set to 0.1, whereas the stiffness of the real object was set to a different value in each experiment (the masses are all set to 5). As a result, the convergence succeeds if the real stiffness is smaller or equal to 0.4.

- The convergence to a local minimum: fig. 8 shows the two position signals of the final learning object and the real object. The learning process has achieved a solution which is not the optimal one, however a rather good one. In the Neural Networks terminology, this situation is called a local minimum, because the learning is "stuck" in a local minimum of the error surface.

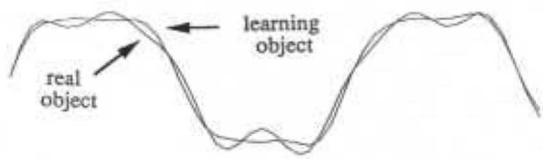

Figure 8: convergence to a local minimum

These results beget some comments. The learning networks succeed in learning a real object, but only when this learning needs small changes of parameters. This is due to the irregular shape of the error surface, which contains many local minima.

To cope with this problem, it is possible to use global minimum tracking algorithms (simulated annealing for example), but these techniques are often very slow. The next direction of study is rather the use of *action*, from the instrumentalist and from the learning object to the real object, in order to achieve a *progressive* learning: thus, the error surface can be more simple at each stage of the learning, since the difficulty is gradual.

5 The cognitive point of view

It is interesting to distinguish the present study from the classical approaches. Generally, intelligent systems are usually split up into two hierarchical levels: the physical level and the control level. The first level is an analogical representation of the environment; the second level controls the first level to learn, plan, predict, act... Consequently, the "intelligence" of such a system is condensed in the control level. Conversely, the learning mechanical networks presented here tend to include an intelligent mechanism directly in the physical level itself.

To apply this to human cognition, it is inconceivable to affirm that intelligence could be reduced to the physical level. However, the learning physical models suggest that the frontier between the intelligent and non-intelligent processes may not be the same as the one between the non-physical and physical representations. Some physical representations, that is analogical representations, may fulfil much more intelligent functions than what it is usually admitted.

6 Conclusion

Analysis in Physical Models may be carried out in quite an elegant way: adding to synthesis networks some properties and elements in order to obtain analysis networks. To that end, Neural Networks provide useful tools.

The first results are encouraging, but the algorithms still need improvements before being implemented in a real-time machine.

References

[Cadoz *et al.*, 1993] Claude Cadoz, Annie Luciani and Jean Loup Florens: CORDIS-ANIMA: a modelling and simulation system for sound and image sysnthesis - the general formalism. *Computer Music Journal*, **17**(1): pp.19-29, 1993.

[William and Peng, 1990] Ronald J. William and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, **2**: pp.490-501, 1990.

[William and Zipser, 1989] Ronald J. William and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**: pp.270-280, 1989.