OVERVIEW OF CONNECTIONIST CONTROL USING MLP*

Cyril Goutte
goutte@laforia.ibp.fr
LAFORIA - Paris VI university
4,place Jussieu - 75252 Paris Cedex 05 - FRANCE

Corinne Ledoux
ledoux@inrets.fr
INRETS - MAIA
Av. Général Malleret Joinville - 94114 Arcueil Cedex - FRANCE

May 23, 1996

Résumé

Mots clefs : Commande de procédés dynamiques, Identification, Réseaux connexionnistes, Perceptrons Multi Couches

Dans ce rapport, nous nous intéressons à l'ensemble des techniques qui ont été proposées dans le cadre de l'identification et de la commande connexionniste. Après une présentation du domaine, nous insistons plus particulièrement sur les points suivants. Dans un premier temps, nous poserons les problèmes relatifs à l'application des réseaux connexionnistes à l'identification et la commande de procédés. Dans un second temps, nous presenterons les caractéristiques et spécificités des techniques d'apprentissage numérique et plus particulièrement des réseaux connexionnistes par rapport aux méthodes classiques. Nous procédons en suite à une classification et à une présentation de ces techniques en quatre catégories : les techniques relatives à l'identification, l'apprentissage du modèle de commande connexionniste grâce á un modèle de commande existant, l'apprentissage direct du modèle de commande connexionniste et enfin l'apprentissage indirect de ce modèle. A la fin de chaque section, nous présentons un ensemble d'applications représentatives des techniques présentées.

Abstract

Keywords : Control of dynamic processes, Identification, Connectionist modelling, Multi-Layer Perceptrons

In this report, we investigate the application of connectionist techniques to identification and process control. After a general presentation, we focus on several aspects, mainly: problems related to the application of connectionist networks to the particular field of identification and control, and characteristics and specificities of numerical learning techniques and connectionist networks, as compared to classical techniques.

We then classify and present these techniques in four categories: first those concerning identification, then connectionist controller learning through an existing controller, direct identification of the neural controller, and indirect learning approaches. In the end of each section we present some explanatory applications of the methods presented.

*Partially supported by the Neuronet Project CEC Network of Excellence N^o 8961

1 Introduction

The purpose of this paper is to provide a structured presentation of the main models introduced in the last years for connectionist control. We have deliberately limited our investigation to techniques using supervised learning. Readers interested in reinforcement learning can refer to [Bar89] or [Lan92].

This overview is obviously biased: it is difficult indeed to produce a simple yet explicit typology convening all proposed architectures. Moreover, it would be virtually infeasible to present a thorough list comprising all models. Thus, our taxonomy is open to criticism, but architectures not specifically presented here can easily be organized to fit within its framework.

We begin by briefly presenting in section 2 the notations used throughout the rest of the paper. Section 3 focuses upon specific aspects which challenge the application of neural networks to the field of control, while section 4 outlines some assets of neural techniques as opposed to traditional methods.

Section 5 briefly presents the connectionist techniques used to model the dynamics of a process. Although not quite a control technique, identification is often a preliminary step towards designing a controller. One will also realise that its applications are numerous and proven.

The classification of architectures used for control results in the following categories:

1. hybrid methods, where the neural network cooperates with a conventional controller, are presented in section 6;

2. techniques which aim at direct learning of the control model with no prior process identification are shown in section 7;

3. lastly, section 8 displays indirect learning of a neural controller through previous identification of the process.

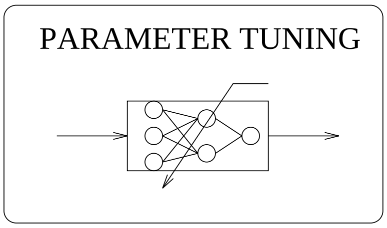

Figure 1: Representation of back-propagation.

2 Notations

In this paper, you will encounter the following notations:

$u(t)$ control signal applied to the process,

$y(t)$ output of the process,

$\hat{u}(t)$ connectionist estimate of the control signal,

$\hat{y}(t)$ connectionist estimate of the output of the process,

$y^*(t)$ output reference signal,

$u^*(t)$ unknown, "ideal" control signal that would produce the reference output,

ϵ_u estimation error on the control signal (which can be $u(t) - \hat{u}(t)$, the squared error, etc),

ϵ_y estimation error on the output signal,

ϵ_y^* reference error on the output signal, i.e. depending on the difference between $y(t)$ and $y^*(t)$. We define $\epsilon_{\hat{y}}^* = \hat{y} - y^*(t)$ in a similar way.

We do not give any general information on either connectionist models, backpropagation or supervised learning. Interested readers can refer to [HKP91]. Figure 1 displays the representation we used for Back-Propagation.

3 Framework and specificities

This report does not restrict the nature of the process we seek to control. It can be *Single Input Single Output*, aka SISO, as well as *Multiple Input Multiple Output*, linear as well as non-linear. Moreover, information made available to the neural network is subject to changes depending on the problem or the kind of network used.

Among all the learning techniques proposed to date, we chose to favor multi-layered perceptrons and gradient back-propagation. The learning strategy -*on-line* or *off-line*- is strongly linked with the dynamics of the process.

Information is made available as input-output couples. The term "input" is to be understood in its broader sense, as it is in fact a vector which possibly contains past values of both control and output signals. Whatever prior knowledge we have of the system is, of course, welcome to influence both the structure and the characteristics of our model.

3.1 Information vacuum

Accordingly, teaching a connectionist model to control a process would consist of estimating couples $(y^*(t), u^*(t))$ with the help of a number of examples. Let us consider that we want such couples available for a given process. This requires a way of obtaining the "perfect" control signal which would allow us to reach a given reference. It is easy to see that presenting the problem as such is awkward: the required training sample could only be obtained with an already existing controller. In other words, we would need the control problem solved in order to start learning our neural controller.

The whole problem of learning a control model is to estimate the error signal without having the desired control signal available. This peculiar situation, quite uncommon in usual neural computation, is examplified on figure 2.

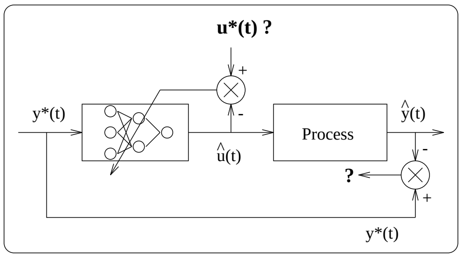

Figure 2: Lack of available information.

Many researchers have then raised the following question: how can we get the error signal we need to learn a control model? Let us quote a few lines from [Bar89] expressing this problem:

> Suppose one wants a connectionist network to learn to control a plant that is too complex, or about which too little is known, to permit the use of conventional techniques for designing controllers. In a typical control problem, one may have target plant outputs but not target network outputs which in [figure 2] are the control signals. How can training signals be provided to the network unless some agency already knows a great deal about how to control the plant?

Indeed, the error signal cannot be taken at the output of the process, but rather at its input. It has been noted, as in [Bar89], that this problem bears certain likenesses with the task of learning the hidden units of a multi-layered network. In both cases, the only reliable error signal lies at the end of the processing, and we seek to grasp the mutual influence of the output of the process (resp. output of the network) on the control signal (resp. hidden cells). Most of the architectures presented below try to overcome this lack of information during the learning of a neural control model.

3.2 Learning problems

This section tries to present those specificities of control related to some aspects of learning. Paragraph 3.2.1 focuses on the algorithm(s) used to optimize the fit criterion. Meanwhile paragraph 3.2.2 concentrates on aspects linked to the learning strategy, like the handling of time.

3.2.1 Optimisation algorithms

As in most other applications of supervised learning, neural control uses gradient descent to optimize a fit criterion computed on empirical data. The observed empirical error depends on the difference between the connectionist model and a sample corresponding to normal behavior of the system. A cost function depending on the model's parameters (i.e. weights of the network) is then computed employing this error. Suitable values of the weights are learned by using some conventional optimization technique to minimize this cost, e.g. stochastic gradient descent and conjugate gradient.

Neuro-control researchers tend to favor the latter, in its most recent versions, as in [Møl93]. Identification and control problems are indeed mainly low dimensional problems. Conjugate gradient algorithms, being more straightforward, perform better on this kind of problem. On the other hand, stochastic gradient descent is less attractive, as its ability of exploring the parameter space is less crucial for these problems.

On the other hand, some control models (e.g. adaptive controllers) require on-line weight update. Though the usual learning scheme can be adapted to handle on-line learning with conjugate gradient, it is often considered more convenient to use stochastic gradient descent whenever on-line learning is necessary.

3.2.2 Strategy

Apart from the optimization algorithm, an accurate choice of learning strategy can result in impressive differences in learning. A fairly theoretical approach to this aspect was presented in [NRRP+93], though unusual handling of notations in this article makes its conclusions less straightforward.

On a practical point of view, let us first recall that in the field of control, the examples feature some kind of time ordering. We then denote by $x(t)$ and $y(t+1)$ the input and output of our system. A simple and artificial way of handling time is to include past values of the output $y(t-i)$ in the input $x(t)$. This was already mentioned at the beginning of 3. On more advanced systems, the use of recurrent connections can be of some help, but is generaly harder to handle.

With $x(0)$ as the starting point, let us denote by $F_w(x(t))$ the mapping carried out by the connectionist model, and by $F_w^N(x(t))$ the same mapping iterated N times. $F_w^T(x(0))$ is then the output value after T times steps. Let us also put forward two cases depending on whether only the goal after T time steps is available or the full trajectory is at our disposal. In the first case, our learning samples are $(x_i(0), y_i(T))$ couples, while in the other, we have the complete data sequences $(x_i(t-1), y_i(t))$, i being an index on the sequences, and t on the time steps in one sequence.

When only the goal after T time steps is available[a], the cost function can be the following:

$$C(w) = \sum_{i=1}^{P} \|y_i(T) - F_w^T(x_i(0))\|^2 \quad (1)$$

When the whole trajectory is known, the simplest learning strategy consists in trying to achieve a good one-step-ahead prediction, by

[a] T can very well be a moving value depending on the goal.

trying to map $x_i(t)$ on $y_i(t+1)$. A suitable fit criterion for such a purpose can be:

$$C(w) = \sum_{i=1}^{P} \sum_{i=1}^{T} ||y_i(t) - F_w(x_i(t-1))||^2 \quad (2)$$

This simple scheme, however, does not actually take into account the order of our $(x_i(t-1), y_i(t))$ examples, which might as well be taken from a sample (x_j, y_j) of size $P \times T$. A better design of the cost function then leads us to:

$$C(w) = \sum_{i=1}^{P} \sum_{t=t_0}^{T} ||y_i(t) - F_w^t(x_i(0))||^2 \quad (3)$$

One must not forget that backpropagating such cost functions as (1) or (3) can be slightly trickier than in the straightforward case.

3.3 Output accuracy

In many neural applications such as classification, the aim is far from perfection. It is often an accepted fact that a certain level of incorrect answers can be tolerated as long as the overall performance reaches a certain level. Furthermore, the output of the system does not have to stick to the theoretical value of the class. For example, if we seek classification in two classes +1 and -1, any positive output will be considered classified in the first class, even though it can actually be quite far from its nominal value.

On the other hand, identification and control of a system can not accept such poor estimates. In this aspect, they are close to function approximation problems, requires far greater accuracy of the outputs values. As we shall see in the following paragraph, the more iterations we intend to perform, the more accurate the model has to be.

3.4 Model robustness

As outlined earlier, controlling a process involves handling time-varying data. Whenever a neural model provides an estimate, it has to be accurate, but it is also extremely important that the estimation error has the smallest possible influence on future estimations.

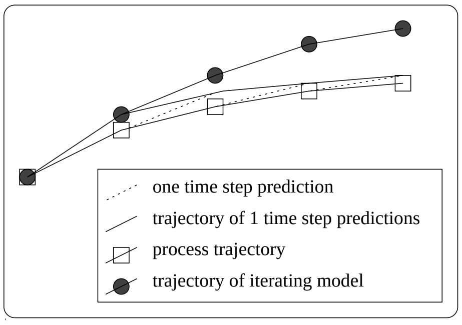

Figure 3: Robustness problem in sytem identification and control.

For system identification, it means that whenever the model iterates, the trajectory error has to be kept within reasonable bounds. This problem is displayed on figure 3 for better understanding.

Robustness is even more crucial for control, as the error on the estimated control signal can be amplified by going through the system. It then affects the output signal which later is to be re-introduced into the controller input.

This mixed accuracy/robustness problem is similar to one that one can meet in speech recognition when no contextual information is used. A fairly good phoneme recognition, let's say around 85%, results in poor recognition of words, of approximately 20% (see [WD91]). This exponential damaging of performance leads us to require more accurate performance from the model as the number of iterations grows.

4 Neural control of non-linear dynamic processes

Within the dynamic systems theory, linear systems have been broadly studied and understood. Numerous results have been obtained thanks to linear algebra, complex variable theory and ordinary linear differential equations, that guarantee stability, robustness and a good dynamic behaviour of such systems.

Conventional modelling/control techniques usually require a preliminary step, during which a mathematical model of the process is designed. This mathematical model is often obtained by approximating the relationship between the input and the observed output of the system. This step involves simplifying hypotheses and considerable knowledge on the process dynamics.

Nonlinear systems theory is currently rather limited compared to the linear one; conventional techniques tend to fail when the system exhibits strong non-linearities. This failure can be explained by the lack of prior knowledge necessary to design the mathematical model.

Whenever the amount of prior knowledge is insufficient to design a mathematical model of the process, learning control offers a convenient alternative. A learning control system is capable of :

- gathering and handling empirical data,
- storing and retrieving experimental information,
- adapting built-up knowledge to integrate new experience.

Further details on learning control systems can be found in chapter 2 of [WS92]. Among those learning techniques, neural networks exhibit interesting properties, that make them good options:

- They provide a mathematical model based on empirical data.

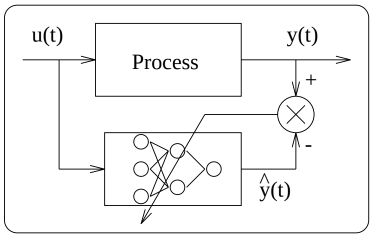

Figure 4: Direct modelling of a process.

- They are universal approximators (cf.[HSW89]) and can hence theoretically model highly complex, strongly non-linear systems.
- They can be trained off-line and adapted on-line to adjust to time-varying dynamics.
- They can handle multi-variate systems.
- They are noise resistant.
- They are suited for real time applications.
- They are easily integrated in hardware.

The problems of modelling and controlling a non-linear second order electromechanical system with varying time constants and saturation effects were convincingly faced in [MSN92]. The first part outlines the limits of the conventional control techniques. The second part justifies the use of neural learning techniques.

5 Dynamics modelling

In forward (or direct) modelling, the neural network is trained to learn the plant's behaviour, as shown in Fig. 4.

5.1 Forward identification

The input supplied to the neural network, $u(t)$, is the input of the plant. The desired output of the model is the plant's output. The signal ϵ_y is the error, based on the difference between desired and observed output. It is back-propagated through the network in order to estimate the synaptic weights. [NP90] presented an overall methodology for identification of non-linear dynamical processes.

Once the neural network has been efficiently trained, it becomes a mathematical model of the system. [ALBC93] identified in this manner kinetic parameters of a chemical reaction. Such a model can be used for :

- designing controllers,
- carrying out simulations,
- diagnosis and long term prediction,
- complementing conventional techniques.

5.2 Applications

Connectionist models have been extensively used for simulation series and tests. [GHB88] modelled a batch fermentation process using multi-layered perceptrons. [PPS91] identified the forward dynamics of a process in noisy conditions, and compared it with results obtained with more conventional techniques.

[LCTb91] presented an application of direct modelling of lactic fermentation to long term prediction. Similarily, [CTCC91] compared the results provided by neural networks and polynomial models to predict the performance of alcoholic fermentation.

As an illustration of the last type of application, [HH90] used a MLP complementary to linear identification technique to model the non-linearities of a process.

6 Hybrid modelling

This section covers some control learning techniques in which modelling is achieved by using a conventional controller (e.g. PID). This

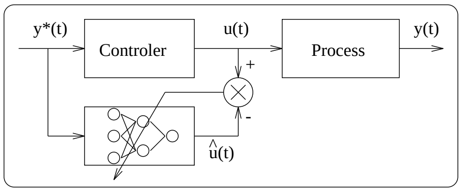

Figure 5: Copy of an existing controller.

conventional controller (CC) is used to generate the missing error signal, according to section 3.1. This can be done in two different ways :

1. the control signal provided by the conventional controller is handled as the missing reference. That is the subject of section 6.1.
2. the control law provided by the conventional controller is used directly as the error signal. That is described further in section 6.2.

6.1 Conventional controller copy

The first approach consists in mimicking the behaviour of the conventional controller. The CC is taken as a particular process—it provides an output for any input it is given. Thus this technique is very similar to forward identification.

The desired output $y^*(t)$, is supplied both to the neural network and the CC to be identified, as shown on figure 5. The output $u(t)$, is then used both to run the process and to compute the error signal. It should be clear that training is based on the approximation that $u(t) \approx u^*(t)$. Obviously this results more in modelling an existing controller than in elaborating a real neuro-control law. Therefore, the purpose of this technique may not be clear at first glance.

Indeed, it may be useful to get a mathematical model of the controller for many reasons.

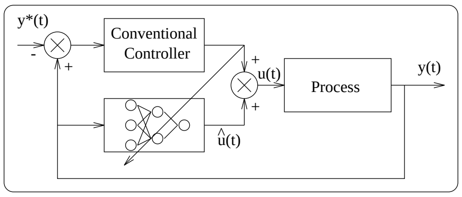

Figure 6: Inverse dynamics model learning.

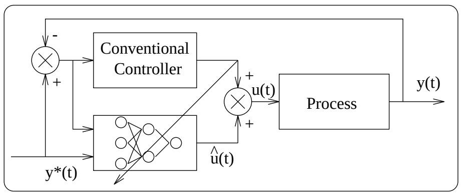

Figure 7: Non-linear regulator learning.

Firstly, the neural model can be simpler and less expensive to handle than the process itself. Secondly, it can be further improved by one of the other methods presented in this survey. Lastly, the intrinsic assets of those models, such as adaptivity and noise resistance, can be used.

6.2 Improving a controller

The key idea of the architectures presented here is the cooperation of neural models with conventional controllers. They were introduced by [GK93] for controlling a two link manipulator. In both methods presented below, the neural model and the conventional feedback controller (CFC) are running simultaneously. The control signal is then the sum of both output signals.

6.2.1 Inverse dynamics model learning

Instead of training a neural network to model the forward dynamics of the plant, it can be trained to model its inverse dynamics. The architecture for such purpose is displayed on figure 6. The output of the process is supplied to the neural model's input. The network's output then participates in the elaboration of the control law $u(t)$ provided to the process.

The output of the CFC depends on the difference between the desired output and the observed output of the process. It is also used as the error signal used to adjust the synaptic weights of the network. As the quality of the control signal increases (thanks to the neural model), the CFC's part in the control signal fades away. The neural network eventually learns how to model the inverse dynamics of the process, i.e. the relationship between $y(t)$ and $u(t)$. Therefore, no retraining has to be performed whenever the conventional controller needs to be changed.

6.2.2 Non-linear regulator learning

Whereas the objective of the previous architecture is the modelling of the inverse dynamics, the purpose here is to regulate the process.

The error signal ϵ_y^* is supplied both to the CFC and to the neural model, which also gets the output reference signal. The training of the network is performed by back propagating the output signal of the CFC as illustrated on Fig. 7. Once the regulation performed is consistent, the classical controller's output is reduced to zero, leaving the regulation task to the neural model alone.

6.3 Applications

Modelling an existing controller is often a preliminary step, before improvement is sought by other, more advanced techniques. For example, [BDL+90] used a PID regulator in order to design a neural controller of a simple non-linear process, and improved it in a second time. [JS90] applied the same methodology to control an aircraft flare during touchdown. [GBK90] used the same principle to

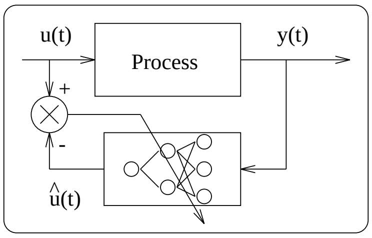

Figure 8: Direct inverse modelling.

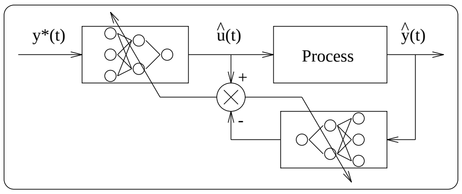

Figure 9: Indirect Learning Architecture.

learn the dynamics of a two link manipulator. [SKML91] operated in two steps to control a distillation column: an existing regulator was first emulated, then optimized thanks to a second network, in charge of estimating specific parameters used by the neural emulator. [GK93], already mentionned, also implemented series of tests, based on the benchmark problem of inverted pendulum. In [SSHR91], the key idea of cooperation is exploited right from the beginning: information provided by a PID is used to initialise the weights.

7 Direct modelling

The control architectures detailed in this section do not require any identification of the process as a preliminary step.

7.1 Direct inverse modelling

The structure presented here is a dual version of the one presented in section 5. Only the functioning of the neural network is inverted. This is why we refer to it as "inverse dynamics modelling".

It runs as follows : an input $u(t)$ is provided to the process, which produces an output $y(t)$. This output is supplied to the neural model, which in turn generates an estimate control signal $\hat{u}(t)$. $y(t)$ is then the reference output corresponding to $u(t)$, which identifies with the reference control signal leading to $y(t)$. The error signal is computed on the basis of the difference between $u(t)$ and $\hat{u}(t)$.

The limits of this architecture, shown in figure 8, are easily perceived. A controller is obviously required to run the process during the training. The behaviour of the neural model will then strongly depend upon the controller used. In particular, the training will only be valid within the domain in which the controller worked.

Moreover, the basic assumption is that the dynamics of the process is effectively invertible. Training is based on the minimisation of the error signal ϵ_u, so there is no guarantee the output error signal ϵ_y^* be minimised as well. The problems raised by this architecture are analyzed in [Kar91] and [Wid88].

7.2 Indirect architecture

This approach corresponds to a particular implementation of the previous architecture where the model being trained is also used to control the process. The desired output $y^*(t)$ is provided to the network which outputs an estimate control signal $\hat{u}(t)$. Once applied to the process, it leads to the actual output $y(t)$. This output is provided to the neural controller anew, resulting in a new estimate $\hat{U}(t)$. The difference between $\hat{u}(t)$ and $\hat{U}(t)$ is then used as the error signal. This architecture is presented in [PSY87] and displayed in figure 9. We display a copy of the neural controller for convenience.

The underlying idea is that driving the control error signal $\hat{U}(t) - \hat{u}(t)$ to zero also can-

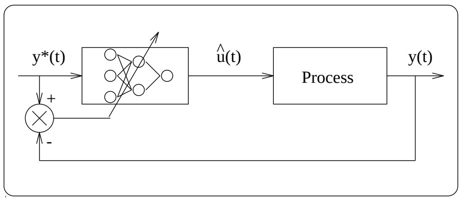

Figure 10: Direct controller learning.

cels the output error signal $y^*(t) - \hat{y}$. However, simulations show that this architecture is not very efficient. A null control error signal does not necessarily lead to a null output error signal.

7.3 Direct learning architecture

In this section, the training of the neural model is performed on the basis of the output error signal ϵ_y^*. The resulting architecture is displayed on figure 10, and was studied in [YY90].

As mentioned earlier in section 3.1, a proper use of back-propagation would require to back-propagate the error signal taken at the output of the neural net. This quantity being unavailable, replacing it by ϵ_y^* can be an extremely rough estimate. This can be justified by the fact that a process is locally linear. The estimate can then be handled by an adequate adaptation of the ϵ parameter in stochastic gradient descent, as long as non-linearities are weak.

In a broader context, simulations show that this architecture is not efficient. Moreover, because of the use of the neural model right after the random initialisation, the robustness of the process plays a great role in the quality of the signals obtained.

7.4 Applications

Inverse dynamics modelling is often used to design feed forward controllers in open loop configuration. Structures presented in this section are widely applied in robotics. Let us just mention [SA90] and [vdS91].

In a more theoretical approach, [VSP9X] made a comparative study of the main forward identification techniques. The comparison is based on the quadratic criteria optimised during training. A variant of direct inverse modelling is proposed and applied to the control of the nonlinear dynamics of an underwater vehicle.

A variant of 7.3, [HB90] showed that neural models can perform direct adaptive control of low order discrete or continuous processes. [YY90] compared neural and adaptive controllers, dealing with the stability of neural structures, with respect to the models considered and the training algorithms implemented.

8 Indirect modelling

This section deals with architectures which use a neural model of the process in order to improve the training of the controller.

8.1 Implementation

As a first step, a neural network is trained to model the dynamics of the process (see section 5). This mathematical model M maps every input $u(t)$ to an output $\hat{y(t)} = M(u(t))$. When a suitable accuracy has been reached (i.e. $\hat{y(t)} \approx y(t)$) the network can be used to train the neural controller.

The key idea lies in the use of the characteristics of back-propagation. The neural network behaves as a differentiable model of the process. The input error can then be estimated by back-propagating the output error through the network.

The implementation of such architecture, proposed in [Jor86] and [Jor89], is depicted in Fig. 11.

The output error is computed using the difference between the actual output and the ref-

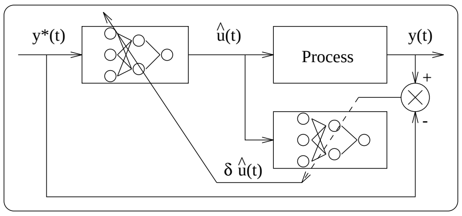

Figure 11: Indirect controller learning.

erence signal. The neural model being sufficiently accurate, the above approximation holds. We then approximate further by stating that :

$$\frac{\partial \epsilon_y^*}{\partial y} \approx \frac{\partial \epsilon_{\hat{y}}^*}{\partial \hat{y}}$$

The missing quantity during training of the neural controller is $\frac{\partial \epsilon_{\hat{u}}^*}{\partial \hat{u}}$, as mentioned in section 3.1. It is easy to see that backpropagating $\frac{\partial \epsilon_{\hat{y}}^*}{\partial \hat{y}}$ through the model then allows us to estimate this missing information.

Notice that in most cases, it is not possible to run the process itself during training. It is then replaced by a mathematical model, which can very well be the neural model. In this case though, the error signal is computed right at the output of the network. However, this architecture is likely to lead to convergence problems. To avoid such troubles, it is convenient to introduce a regularization term during training.

8.2 Controller adaptation

In figure 11, we did not put the model output to any use. The network is used there mainly for its ability to be derived by backpropagation. Nevertheless this information can be exploited to perform an on-line adaptation of the model. It is reassuring to notice that the resulting architecture is very similar to adaptive control schemes.

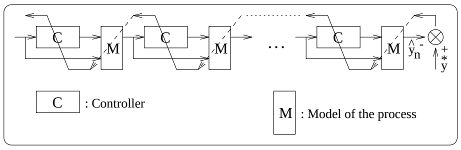

Figure 12: Indirect learning through time.

8.3 Related architectures

The key concept of the architecture proposed by [NW89] is similar to the one of section 8.1. It consists, more or less, in iterating Jordan's architecture over several time steps. As a matter of fact, both architectures are quite similar. The main difference lies in the training algorithm used rather than in the architecture itself. An idea of this difference has been presented in section 3.2.2, though it is not the purpose of this paper to present the various training methods existing. It is displayed on figure 12.

9 Conclusion

The architectures presented above constitute a more or less thorough overview of the most widely used neuro-control techniques. Architectures not mentionned here should be easy to relate to one of the schemes described.

This paper was written with the following aim : to provide connectionists and automatists with alternative neuro-control schemes, suited for solving a given problem. Ordering the architectures was an additional difficulty. The classification follows an increasing difficulty of implementation, as well as efficiency. This explains why the schemes detailed in the last section seem to get the researcher's favor.

Being fully objective would require implementing all the models presented and applying them either to benchmark and to real applications. Such a work would conclude on their abilities to control the process, their effectiveness and the difficulties faced during their implementation. The apparent simplicity of such a task hides crucial choices, inherent to any comparative study. Benchmark problems have to be defined. They have to characterise a particular class of real problems. They also have to be general enough in order not to favor one or the other architecture. As a general matter, each of the schemes proposed is dedicated to its author's particular application.

References

[ALBC93] G. Acuna, E. Latrille, C. Beal, and G. Corrieu. Modélisation par réseaux de neurones des cinétiques réactionnelles caractéristiques des fermentations lactiques batch à ph régulé. In *4ème congrès Français de Génie des Procédés*, Grenoble, 1993.

[Bar89] A.G. Barto. Connectionist learning for control: An overview. Technical Report 89-89, COINS, September 1989.

[BDL+90] H. Bleuler, D. Diez, G. Lauber, U. Meyer, and D. Zlatnik. Nonlinear neural network control with application example. In *International Neural Networks Conference*, volume I, pages 201–204, Paris, 1990.

[CTCC91] Y. Cleran, J. Thibault, A. Cheruy, and G. Corrieu. Comparison of prediction performances between models obtained by the group method of data handling and neural networks for the alcoholic fermentation rate in enology. *Journal of Fermentation and Bioengineering*, 71(5):356–362, 1991.

[GBK90] A. Guez and I. Bar-Kana. Two degree of freedom robot adaptative controller. In *American Conference on Control*, pages 3001–3006, San Diego, May 1990.

[GHB88] S. Gehlen, M. Hormel, and S. Bohrer. A learning control scheme with neuron-like associative memories for the control of biotechnological processes. In *Neuro Nimes*, pages 113–126, 1988.

[GK93] H. Gomi and M. Kawato. Neural network control for a closed loop system using feedback error learning. *Neural Networks*, VI:933–946, 1993.

[HB90] Helferty and Biswas. Neuromorphic control as a self-tuning regulator. In *IEEE International Symposium on Intelligent Control*, 1990.

[HH90] D. Haesloop and B.R. Holt. A neural network structure for system identification. In *American Conference on Control*, pages 2460–2465, San Diego, May 1990.

[HKP91] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the theory of neural computation.* Addison-Wesley, 1991.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 1989.

[Jor86] M. Jordan. Serial order: A paralel distributed processing approach. Technical Report 8604, Institute for Cognitive Science Report, 1986.

[Jor89] M. Jordan. Generic constraints on underspecified target trajectories. In *International Joint Conference on Neural Networks*, volume I, pages 217–225, 1989.

[JS90] C.C. Jorgensen and C. Schley. A neural network baseline problem for control of aircraft flare and touchdown. *Neural networks for Control*, pages 403–425, 1990.

[Kar91] G. Karsai. learning to control : some practical experiments with neural networks. In *International Joint Conference on Neural Networks*, volume II, pages 701–707, Seattle, WA, July 1991.

[Lan92] T. Langlois. *Algorithmes d'apprentissage par renforcement pour la commande adaptative.* PhD thesis, Université Téchnologique de Compiègne, 1992.

[LCTb91] E. Latrille, G. Corrieu, and J. Thibault. ph predictions and final fermentation time determination in lactic acid batch fermentations. In *European Symposium on Computer Aided Process Engineering*, volume 2, 1991.

[Møl93] M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

[MSN92] R. Murray-Smith and D. Neumerkel. Neural networks for modelling and control of a non linear dynamic system. In *IEEE-Symposium on Intelligent Control*, pages 404–409, 1992.

[NP90] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27, 1990.

[NRRP+93] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos. Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms. *Neural Networks*, 5(2):165–199, 1993.

[NW89] D. Nguyen and B. Widrow. The truck backer-upper: an example of self-learning in neural networks. In *International Joint Conference on Neural Networks*, volume II, pages 357–363, June 1989.

[PPS91] Y.H Pao, G.H. Park, and D.J. Sobajic. System identification and noise cancellation : a quantitative comparative study of the kalman filtering and neural net approaches. In *American Conference on Control*, pages 1408–1411, Boston, MA, June 1991.

[PSY87] D. Psaltis, A. Sideris, and A. Yamamura. Neural controllers. In *International Neural Networks Conference*, volume IV, pages 551–558, 1987.

[SA90] R.M. Sanner and D.L. Akin. Neuromorphic pitch attitude regulation of an underwater telerobot. *IEEE Transactions on Systems, Man and Cybernetics*, 10:62–68, April 1990.

[SKML91] J. Steck, K. Krishnamurthy, B. McMillin, and G. Leininger. Neural modeling and control of a distillation column. In *International Joint Conference on Neu-*

ral Networks, volume II, pages 771–773, Seattle, WA, July 1991.

[SSHR91] G.M. Scott, J.W. Shavlik, and W. Harmon Ray. Refining pid controllers using neural networks. In *NIPS 4*, pages 555–562. Advances in Neural Information Processing Systems, 1991.

[vdS91] P.P. van der Smagt. A real time learning neural robot controller. In *International Conference on Artificial Neural Networks*, pages 351–356, Espoo, Finland, June 1991.

[VSP9X] K.P. Venugopal, R. Sudhakar, and A.S. Pandya. An improved scheme for direct adaptive control of dynamical systems using backpropagation neural networks. Article, 199X.

[WD91] D. Wettschereck and T. Dietterich. Improving the performance of radial basis function networks by learning center locations. In *NIPS 4*, volume 4, pages 1133–1140. Advances in Neural Information Processing Systems, 1991.

[Wid88] B. Widrow. Adaptive inverse control. In *Proceedings of the IFAC Adaptive Systems in Control and Signal Processing Conference*, pages 1–5, Lund, 1988.

[WS92] D.A White and D.A Sofge. *Handbook of Intelligen Control.* VNR Computer Library, 1992.

[YY90] Yabuta and Yamada. Possibility of neural networks controller for robot manipulators. In *IEEE International Conference on Robotics and Automation*, 1990.