Some Preliminary Comparisons
Between a Neural Adaptive Controller
and a Model Reference Adaptive Controller

Marco Saerens¹, Alain Soquet², Jean-Michel Renders,
and Hugues Bersini¹

Laboratoire IRIDIA
Université Libre de Bruxelles
Av. F. Roosevelt, 50 – CP 194/6
B-1050 Bruxelles – Belgium

Abstract

Our aim in this paper is to make some preliminary
comparisons between a neural adaptive controller and a
Model Reference Adaptive Controller, on a simple and well
defined first-order problem. We focus on the rate of
convergence, and on the capacity to control non-linear and
time-varying systems. Results from a first experiment
show that the MRAC always converges faster and performs
better for linear systems, but that its performances decline
in case of non-linearity: the more abrupt the non-linearity,
the stronger the decline in performance. On the contrary,
this phenomenon is not observed for the neural net, whose
performances do not vary significantly when the plant
changes from linear to non-linear. Results from a second

¹ The following text presents research results of the Belgian National incentive-program
for fundamental research in artificial intelligence initiated by the Belgian State, Prime
Minister’s Office, Science Policy Programming. The scientific responsibility is assumed
by the authors.
experiment show that the neural net adapts his parameters well for fast time-varying processes. This shows that a neural network, although it converges much more slowly, could be more appropriate in the case of control of unknown non-linear processes. However, we must stress that much has to be done to study the conditions of convergence of networks, in order to understand what can be really done by neural nets in non-linear control.

Introduction

Several approaches are possible when using neural nets for the control of plants (Psaltis, Sideris & Yamamura, 1987; Barto, 1990). The specialized learning approach fits in perfectly with the classical adaptive control attitude. In this case, the neural controller uses the difference between the actual output of the plant and the desired output to directly adapt the weights of connections. However, to make specialized learning possible, we need some prior knowledge on the way the plant reacts to slight control modifications, i.e. the Jacobian of the plant (see back-propagation through the plant below). One possible strategy consists in approximating the partial derivatives by plotting the plant reactions to slight control modifications at the operating points (Psaltis, Sideris & Yamamura, 1987). A more direct solution is to approximate the unknown derivatives by constant values (Sarens & Soquet, 1989), or to estimate them by a linear model of the plant (Sarens & Soquet, 1991). This simple adaptive control approach is the one explained and used in this paper. A different approach is to apply Lyapunov's stability theory to derive adaptation laws which guarantee the stability of the training process ( Renders, 1990). This algorithm was applied to robot's trajectory control.

Another recent and sophisticated technique has been proposed (Jordan, 1989; Jordan & Rumelhart, 1991; Nguyen & Widrow, 1989, 1990), which incorporates the default prior knowledge directly in a network and links the neural controller to a neural emulator of the plant - a neural network that identifies the process. As pointed out by Narendra & Parthasarathy (Narendra & Parthasarathy, 1990), it corresponds to the classical indirect control approach. In indirect control, the parameters of the plant are estimated at any instant k, and the parameters of the controller (in this case, the weights of the neural controller) are adjusted assuming that the estimated parameters of the plant represent the true values. This technique makes the control quite flexible for indeterminate learning problems, allowing generic constraints to be expressed separately at the control and at the behavioural level. However it demands, in compensation, either a preceding learning stage (the identification of the plant) or a "neurally expressed" prior knowledge of the dynamics of the plant. Narendra & Parthasarathy (1990) use a similar indirect technique to show that identification and adaptive control schemes are practically feasible.
Indeed, Narendra & Parthasarathy (1990) showed by simulations that a neural network can be used effectively for the identification and control of nonlinear dynamical processes. The results are based on the universal approximation properties of three-layer neural networks. As a matter of fact, it has been shown (Hornik, Stinchombe & White, 1989 and 1990; Stinchombe & White, 1989; for a review, see White, 1989) that a three-layer network (one hidden layer) with an arbitrarily large number of nodes in the hidden layer can approximate any continuous function \( \mathbb{R}^n \rightarrow \mathbb{R}^m \) over a compact subset of \( \mathbb{R}^n \). This provides the motivation to study neural networks in the framework of nonlinear systems theory (see Narendra & Parthasarathy, 1990, for more details). However, the choice of identification and controller models for nonlinear plants is a formidable problem and successful identification and control has to depend upon several strong assumptions regarding the input-output behaviour of the plant. Complete controllability and observability must be assumed (see Narendra & Parthasarathy, 1990, for a discussion). They point out that considerable progress in nonlinear control theory will be needed to obtain rigorous solutions to the identification and control problems.

Our aim in this paper is to make some preliminary comparisons between a neural adaptive controller and a Model Reference Adaptive Controller, on a simple and well defined first-order problem. We focus on the rate of convergence, and on the capacity to control non-linear and time-varying systems.

**Control using backpropagation**

We will briefly describe four different architectures and learning schemes that can be used to control a plant by using backpropagation (Psaltis, Sideris & Yamamura, 1987). Then, we will focus on specialized learning, for which the most interesting results have been obtained.

![Diagram](image-url)

*Figure 1: General learning architecture.*

**General learning architecture:** In the case of general learning, the training stage consists in using the plant to produce a set of input-output pairs. These pairs are then used as patterns for training the neural network. Thus, during the training stage,
the control parameters $u_x$ are chosen randomly within their working range. Those parameters are then injected into the plant which supplies output values $y_\beta$. The back-propagation algorithm is used to train the network to relate the outputs of the network $u_x$ to the inputs $y_\beta$ (Figure 1). After this learning stage, the controller is able to provide the correct control parameters to reach any desired target. This method has several drawbacks: first, learning must be performed off-line; second, the network cannot limit its working range to the $y_\beta$ that are actually relevant; and finally, control is not adaptive. However, general learning is very popular; for instance, Kuperstein uses this kind of architecture to learn sensory-motor coordination from experience (Kuperstein, 1988; Kuperstein & Rubinstein, 1988).

Indirect learning with plant identification: The idea is to identify the plant with a neural network (Figure 2). Then if $y_a(k+1) = g_a[y_\beta(k), \ldots, y_\beta(k-p+1), u_\beta(k), \ldots, u_\beta(k-p+l)]$ is the function identified by the net, the optimal control parameters $u_\beta(k)$ are computed at each step $k$ by solving:

$$0 = y_a^{d}(k+1) - g_a[y_\beta(k), \ldots, y_\beta(k-p+1), u_\beta(k), \ldots, u_\beta(k-p+l)]$$

where the $y_a^{d}(k)$ are the desired outputs, by using a non-linear optimization technique. This algorithm is used by Ydstie (1990) and by Bhat & McAvoy (1990) for the control of simulated nonlinear chemical processes. However, it needs important computing power because of the optimization process that is performed at each iteration.

![Diagram](image)

**Figure 2: Indirect learning with plant identification.**

Indirect learning architecture: In this case, the desired target ($y_a^{d}$) propagates through the net, producing the plant input $u_x$. The response of the plant to this input is then used as the input to a copy of the controlling network (Figure 3). The difference between the output of the network and its copy is the error signal used to correct the weights. However, simulations show that the network can settle to a solution that maps all the desired targets to a single plant input, which gives zero training error, but obviously a non-zero total error (Psaltis, Sideris & Yamamura,
Learning is indirect because the difference between the output of the plant and the desired output are not directly reduced.

Specialized learning architecture (closed-loop learning): Psaltis, Sideris & Yamamura propose another learning scheme called specialized learning. Specialized learning differs from general and indirect learning by the fact that the controller learns no longer from input-output pairs but from a direct evaluation of the network accuracy, with respect to the output of the plant (Figure 4). The network uses the difference between the actual output of the plant $y_p$ and the desired output $y_d$ to change the weights of connections. Specialized learning avoids several drawbacks of general learning: there is no longer a specific training stage during which the controller is not operational, and the network learns directly on the domain of relevant $y_p$. Moreover, the controller learns continually, and is therefore adaptive. Yet, the evaluation of the error from the output requires prior knowledge of the plant. We will now investigate in a more technical view the specialized learning architecture, which appears to be the most interesting one.

**Figure 3: Indirect learning architecture**  
**Figure 4: Specialized learning architecture**

Back-propagation through the plant

In the case of our neural controller (Figure 4), the final layer provides at each time step $k$ the inputs $u_{g}(k)$ of the plant, which outputs $y_{g}(k+1)$ and not the desired target $y_d(k+1)$. For a process of order $p$, the neural net is given the $p$ last output vectors of the plant, as well as the $p$ last control parameter vectors, in order to be able to reconstruct the state of the plant. The net must provide the series of control parameters $u_{g}(k-1), u_{g}(k-2), ..., u_{g}(k-p)$ that minimize $E(k) = \frac{1}{2} \sum_{d} (y_{d}(k) - y_{d}'(k))^2$.

This will be done by performing a gradient descent in $E(w_{ab})$, by using back-propagation algorithm (Le Cun, 1985; Rumelhart, Hinton & Williams, 1986). To
compute the gradient, we introduce the lagrangian \( L(y_d(k), u_d(k-i), Y_a(k-i), w_a \delta(k-i)) \) at time \( k \):

\[
L(k) = \frac{1}{2} \sum_{a} (y_a(k) - y_a^d(k))^2
\]  

(1a)

\[+ \sum_{a} \Gamma_a(k) \left( y_a(k) - y_a(k-I), \ldots, y_a(k-p), u_a(k-I), \ldots, u_a(k-p) \right) \]  

(1b)

\[+ \sum_{a} \sum_{i=1}^{p} \lambda_a(k-i) \left( u_a(k-i) - U_a(k-i) \right) \]  

(1c)

\[+ \sum_{i=1}^{p} \sum_{a} \lambda_a^I(k-i) \left( U_a^I(k-i) - \sum_{j=1}^{a} w_a(j-i) U_b^I(k-j) \right) \]  

(1d)

where \( U_a(k-i) \) is the activation of unit \( a \) at layer \( l \) and time step \( (k-i) \) and \( w_a \delta(k-i) \) is the weight of the connection between unit \( a \) at layer \( l \) and unit \( \beta \) at layer \( l-1 \), also at time step \( (k-i) \). The first term of equation (1a) corresponds to the cost function, the second one (1b) to the input-output representation of the process of order \( p \), the third one (1c) imposes that the units of the output layer \( U_a(k) \) (layer \( q \)) provide the control parameters \( u_a(k) \) at each time step \( k \), and the last one (1d) corresponds to the transfer functions of the units. \( \Gamma_a(k), \lambda_a(k), \lambda_a^I(k) \) are lagrange multipliers.

The extremum conditions are \( \frac{\partial L}{\partial y_a(k)} = 0, \frac{\partial L}{\partial u_a(k)} = 0, \frac{\partial L}{\partial U_a^I(k)} = 0 \), and allow us to compute the lagrange multipliers \( \Gamma_a(k), \lambda_a(k), \lambda_a^I(k) \) (Le Cun, 1989). The weights are obtained by performing a gradient descent: \( w_{aq}(k+1) = w_{aq}(k) - \eta \frac{\partial L(k)}{\partial w_{aq}(k)} \). This results in the following rule:

\[
\delta_a^q(j) = f^q(j) \sum_{a} \left( y_a(k) - y_a^d(k) \right) \frac{\partial y_a(k)}{\partial u_a(j)}
\]

where \( \delta_a^q(j) \) is the back-propagated error at the output layer \( q \) and \( f^q(j) \) is the derivative of the sigmoid function. This way, the weights at time \( j \) are modified.
for time step \( k=j+1 \) until \( k=j+p \). By cumulating all these contributions, we obtain for the back-propagated error at the output layer (for more details, see Saerens, 1991; a less rigorous derivation can be found in Saerens & Soquet, 1991):

\[
\delta_f^q(j) = f^q_f(j) \sum_{k=j+1}^{j+p} \sum_{a} (y_a(k) - y_d(k)) \frac{\partial \hat{y}_a(k)}{\partial \xi_a(j)} = f^q_f(j) \sum_{k=j+1}^{j+p} \frac{\partial E(k)}{\partial \xi_a(j)}
\]  

(2)

At this stage, we see that the learning algorithm involves the Jacobian of the plant \( \frac{\partial \hat{y}_a(k)}{\partial \xi_a(j)} \), which is unknown (in Jordan's method, the Jacobian is obtained by back-propagation through a neural network that identifies the plant; Jordan, 1989; Jordan & Rumelhart, 1991). For a \( p \)-order process, we can estimate the different terms of the Jacobian by a linear model (which corresponds to the DARMA form) of the plant:

\[
y_d(k) = \sum_{i=1}^{p} \sum_{p} a_{ij}^l y_d(k-i) + \sum_{i=1}^{p} \sum_{p} b_{ij}^l u_d(k-i)
\]

and the parameters \( a_{ij}^l \) and \( b_{ij}^l \) can be estimated on-line by using, for example, an adaptive least-squares algorithm (see, for instance, Goodwin & Sin, 1984). However, for a first-order process, we can simply use the learning rule:

\[
\delta_f^q(k) = f^q_f(k) \xi_a \text{ sign} \left( \frac{\partial y_a(k+1)}{\partial u_a(k)} \right) (y_d(k+1) - y_d(k+1))
\]

(3)

where \( \xi_a \) is a stochastic variable with mean 1.0, uniformly distributed on \([0.5, 1.5]\) (Saerens, 1991). This approximation performs error minimisation in the following sense: For each term of \( E(k) \), the scalar product between true gradient descent and gradient approximation is positive, which leads to an error decrease. This is the algorithm that will be used for the comparisons.

This method (with simple sign approximation) has been applied to some toy problems, and to the pole-balancing problem (Saerens & Soquet, 1989, 1991). We also used this technique to provide an acoustic-to-articulatory inversion of a vocal tract model, which is a nonlinear many-to-one problem (Soquet, Saerens & Jospa, 1990, 1991ab).

**Model Reference Adaptive Control**

Model-Reference Adaptive Controllers are designed following two main ideas (Figure 5). First, the desired behaviour of the system is specified by a reference model, which gives the desired response to a command signal. Second, the
parameters of the regulator are tuned on the basis of the error between the outputs of the process and the reference model. The control architecture thus consists of two loops: an inner loop, which provides the ordinary control feedback, and an outer loop, which adapts the parameters of the inner loop following some adjustment mechanism.

![Diagram of MRAC architecture](image)

*Figure 5: MRAC architecture.*

The analysis and design of a MRAC can be done by three methods: the gradient approach, the Lyapunov stability theory and the passivity theory (see, for instance, Aström & Wittenmark, 1989). We will consider the gradient approach (the MIT-rule), which is the simplest one, and which is very similar to the back-propagation algorithm.

Assume that we try to change the parameters of the regulator so that the error between the outputs of the process \( y_d(k) \) and the reference model \( y^m_r(k) \) is driven towards zero. This can be done by minimizing the criterion:

\[
J(k) = \frac{1}{2} \sum_{\alpha} (y_d(k) - y^m_r(k))^2
\]

Therefore, we adopt a steepest-descent method, by changing the regulator parameters \( \theta_\beta \) in the direction of the negative gradient of \( J \):

\[
\Delta \theta_\beta = -\eta \frac{\partial J}{\partial \theta_\beta} = -\eta \sum_{\alpha} \frac{\partial y_d(k)}{\partial \theta_\beta} (y_d(k) - y^m_r(k))
\]

This adjustment law is commonly referred as the MIT-rule.
First experiment

Experiment description. The two control methods are applied to a simulated first order problem:

\[ y(k+1) = ay(k) + bf(u(k)) + ce(k) \]  

(4)

where \( u(k) \) is the control variable at step \( k \), \( y(k) \) the measured output, and \( e(k) \) a random noise term, uniformly distributed in \([-0.1, +0.1]\). The parameters of the process are fixed at \( a = 0.1 \) and \( b = 0.9 \). The parameter \( c \) will be set to 0 for a noiseless process and to 1 for a process with noise.

The neural network is composed of two input units, two hidden units and one output unit. It is given the actual state of the process – \( y(k) \) – and the desired output of the plant – \( y^d(k+1) \), and provides the control parameter – \( u(k) \). Every unit of each layer is connected with units of adjacent layers. No effort was made to reduce the dynamic range of the inputs and the outputs: they were scaled to reflect continuous values from \(-1.0\) to \(+1.0\), representing normalized signals.

For the MRAC, the regulator parameters are adjusted following the MIT-rule (see Åström & Wittenmark, 1989, for the proof) by using a linear law (assuming that the dependence between the parameter and the output of the system is unknown).

The desired output of the plant \( y^d(k) \) is a triangular waveform, varying from \(-1\) and \(+1\), with a defined period (Figure 6). The only a priori knowledge of the process being controlled is the sign of \( b \), which is both necessary for the MRAC and for the neural network. The sampling rate was one hertz, and the initial value for \( y(0) \) was 0.

Three groups of simulations are conducted for both controllers. For the three groups, the process being controlled is described by equation (4) with respectively \( f(u) = u \), \( f(u) = \text{sign}(u) u^2 \), and \( f(u) = u^2 \). For each group, results were collected for two different conditions: one with a noise free process (\( c = 0 \)), and one with a noise term (\( c = 1 \)). The aim is to compare the performance and the convergence rate of the two methods in relation with the non-linearity of the process.

Results.

1. Linear case – \( u \)

Figures 7a and 7b show comparisons of convergence rate between the MRAC – lower curve – and the neural network (two different gradient parameters are used for the network: 0.025 – upper curve – and 0.05 – intermediate curve; the momentum term is 0.9). Figures 7a and 7b show results respectively without noise and with noise. The period of the triangular waveform was 30 steps. The error is defined as the difference between the desired and the actual output of the plant, averaged over one period for any run, and averaged further over ten runs, carried out with different initial weights and regulator parameters. We clearly see that the MRAC learns
much more quickly than the neural network in both cases. Moreover, the MRAC performs much better than the neural network in the noiseless case, but less obviously so in the presence of noise. In both cases, the MRAC is perfectly adapting its regulator parameters to produce minimal error.

![Figure 7: linear case (a) without noise, (b) with noise.](image)

2. Non-linear case – $u^2$

Figures 8a and 8b allow comparisons of error between the MRAC – upper curve – and the neural network – lower curve (the gradient parameter is 0.05; the momentum term is 0.9). Figures 8a and 8b show results respectively without noise and with noise.

![Figure 8: $u^2$ (a) without noise, (b) with noise.](image)

The curves plot the error after a learning (or convergence) stage, in function of the period of the triangular waveform. The error is computed after a learning stage of 2000 steps. The error is defined as the difference between the desired and the actual output of the plant, averaged over ten periods for any run, and averaged further over ten runs. The error is plotted for seven periods of the waveform, from 20 to 200 steps. We observe that the neural network now performs better than the MRAC. The neural network error remains fairly stable, while the MRAC error increases with the frequency of the triangular waveform. This can easily be understood: the desired outputs are changing too quickly given the adapting time of the regulator parameters. Moreover, comparing Figures 7 and 8 clearly shows that the performance of the neural net does not vary significantly when the plant changes
Figure 10: Example of evolution of the system, after a convergence time of 2000 steps, for the MRAC (upper graph) and the neural network (lower graph). The values of the output of the plant (normal line) and the desired output (triangular waveform in bold) are represented in terms of time step $k$. The control parameter is also represented, with a shift of $+1.0$ for legibility.
from linear to non-linear. In fact, the neural net simply approximates the non-linear control law $u[y(k), y^2(k+1)]$.

3. Non-linear case — $u^3$

Figure 9ab allows comparisons of error between the MRAC — upper curve — and the neural network — lower curve. The experimental conditions are similar to the first non-linear case. The MRAC decline in performance is now more obvious. An example of the behaviour of both regulators is shown in Figure 6.

![Figure 9: $u^3$ (a) without noise, (b) with noise.](image)

**Second experiment**

**Experiment description.** The two control methods were applied to the same simulate first order problem as in previous experiment:

$$y(k+1) = a(k) y(k) + b u(k) + c e(k)$$  \hfill (5a)

However, in this case, we consider a time-varying process:

$$a(k) = 0.5 + 0.3 \sin \left( \frac{2\pi k}{T} \right)$$  \hfill (5b)

in order to evaluate the capacity of the neural net to adapt its weights on-line. In this experiment, we choose $b = 1.0$. The error is plotted for four periods of the sinusoid, from $T = 20$ to $200$ steps. The period of the triangular waveform (the desired output) is fixed at $75$ steps. The other experimental conditions are exactly the same as in previous experiment (non-linear case).

**Results.** Results are shown in Figure 11ab, which allows comparisons of error between the MRAC — lower curve — and the neural network — upper curve. The MRAC adapts its parameters in a more accurate and rapid way. However the performances of the neural net are quite stable, and not as bad as we could expect (the neural net has to adjust 10 parameters: 6 weights, and 4 thresholds). One
example of the behaviour of both regulators is shown in Figure 10. We observe that the control law is very similar.

Figure 11: Results for a time-varying process, (a) without noise, (b) with noise.

Conclusion

Results from a first experiment show that the MRAC always converges faster and performs better for linear systems, but that its performance declines in case of non-linearity: the more abrupt the non-linearity, the stronger the decline in performance. On the contrary, this phenomenon is not observed for the neural net, whose performances do not vary significantly when the plant changes from linear to non-linear. Results from a second experiment show that the neural net adapts his parameters well for fast time-varying processes. This shows that a neural network, although it converges much more slowly, could be more appropriate in the case of control of unknown non-linear processes.

However, we must stress that much has to be done to study the conditions of convergence of networks, in order to understand what can be really done by neural nets in non-linear control. Indeed, systematic design methods, which guarantee the stability of the adaptation process and the success of the overall design, are still missing. It should also be of interest to study the possibility of controlling simple non-minimal phase problems. More comparisons with classical adaptive techniques should also be performed.
Figure 6: Example of convergence of the MRAC (upper graph) and the neural network (lower graph). The values of the output of the plant (normal line) and the desired output (triangular waveform in bold) are represented in terms of time step $k$. 
References


