Structural and Training Strategies for Direct Neural control Systems

BOUBEKEUR MENDIL[1] & KHIER BENMAHAMMED[2]

[1] Electrical Engineering Department, University of Bejaia, ALGERIA.
Fax : 213 521 4332
[2] Electronics Institute University of Setif , ALGERIA.

Abstract: - This paper presents new structural and training strategies for neural control systems. First, we show that the existing neural control structures can be brought to a single learning structure. Then, a new *FeedForward Error Propagation* (FEP) learning algorithm is tailored especially for training controllers in a direct manner without the use of identifier or any form of the plant model. This avoids the uncertainty and the computational and storage requirements related to the forward model. Finally, The main features of proposed schemes are evaluated using simulation examples.

Key-words: -Adaptive and optimal control, neural networks, training algorithms, FEP algorithm.

1 Unified Direct Learning Control Structure

The key problem in control systems is to design a controller which generates the desired control input so that the behavior of the plant meets a collection of specifications constituting the control objective. Usually, these specifications are expressed in terms of speed, accuracy, and stability. There are mainly, two classes of control problems:

- Tracking problems in which the objective is to follow a desired trajectory or a reference model. The self-tuning regulators (STR) and model reference adaptive control (MRAC) are the mostly used methods in such situations.
- Optimal control problems in which the objective is to extremize a functional of the controlled system's behavior. The key issue is constrained optimization. Dynamic programming is the well known approach for converting dynamic optimization into static ones.

The approach proposed in this paper does not assume any special form for the plant model, but we are rather interested in unknown MIMO nonlinear plants. However, for the sake simplicity and clearness, we use a SISO plant described by the nonlinear autoregressive moving average (NARMA) model [1]

$$y(k+d) = f[Y_n(k), U_m(k)] \qquad (1)$$

where $f : \mathbb{R}^{n+m} \to \mathbb{R}$ is a nonlinear function, $Y_n(k) = [y(k), y(k-1),\ldots, y(k-n+1)]$ is the output sequence, $U_m(k) = [u(k), u(k-1),\ldots, y(k-m+1)]$ is the input sequence, and d denotes the relative degree which represents the pure delay from input u to output y (resulted from the inertia of the physical plant). This notion of relative degree will be used later to explain some training problems of neural controllers.

If the control problem is expressed in terms of trajectory tracking, then the aim is to determine control law of the form $u(k)=g_1[Y_n(k), y_r(k+d), U_m(k)]$ so that the error $e_c(k+d)$ at the future instant $(k+d)$, between the plant output $y(k+d)$ and the desired trajectory $y_r(k+d)$, becomes small. This problem is usually addressed by the STR approach. However, in the MRAC the control law $u(k)=g_2[Y_n(k), r(k+), U_m(k)]$ is to be found so that the plant output matches or tracks as closely as possible a reference model output. Here $r(k)$ denotes the reference input signal. In the second family of control problems, the aim is to determine a control law $u(k)=g_3[Y_n(k), U_m(k)]$ which optimizes some performance criterion.

If a neural network is used to implement the nonlinear control map $g_i(.)$, then the key issue is to find the convenient way to adjust the controller network weights. The main difficulty here is that of credit assignment, that is how the error in the plant output should be used to modify the controller since the unknown physical plant lies between the controller output and the plant output.

Various strategies have been tried to overcome this problem. The simplest solution is to train a controller to imitate a knowledgeable controller (*i.e.*, a teacher) which already knows how to control the plant. This supervisor can be a human expert or an artificial expert *(e*.g., a rule base) as proposed in [2][3]. Yet, this solution is not often practical.

The second approach, referred to as generalized inverse modeling [4], is to train directly a neural network to learn the inverse dynamics of the plant and then to use it as a controller to generate the control signal. Unfortunately, this approach suffers from some drawbacks: the learning procedure is not goal directed and the minimization of the error on the controller output does not guarantee minimization of the overall system error, and in a nonlinear system the mapping is not one-one then an incorrect inverse can be obtained. Even the scheme proposed in [5], based on a single network with a switching logic, inherits the same problems.

Currently, two main approaches are used to tackle the problem of training the controller net. Both of them make use of an extra network to overcome the credit assignment problem stated above.

- *Reinforcement learning* in which a critic network is trained, using one of the adaptive critic methods, to provide reinforcement signals to the action network (*i.e.,* controller) and to assign credit to its individual actions. The critic network output plays the role of an error measure in the update of the action network weights [6][7]. Since reinforcement learning refers only to information of the kind '*right or wrong'* to guide the training process, it does not require the network to generate approximations to the desired detailed input-output mappings as in the case of supervised learning schemes.
- *Forward Modeling approach* which uses a sensitivity model(*i.e.,* a Jacobian or a neural identifier) of the plant to serve as a channel to back-propagate the error at the plant output up to the plant input [4][8][9]. This method is considered as a general purpose control scheme in the neural control literature. However, if little knowledge of the nonlinear plant is available, it is impossible to obtain an analytical expression for the Jacobian. Furthermore, the technique of neural identification can lead to wrong results when the plant involves significant delays [10]. Briefly, the NARMA model (1) means that the input at instant *k* affects the output only *d* units of time later. Therefore, if a neural identifier is used, then the back-propagated error at the controller output will always be zero since only delayed values affect the identifier output and hence no weight updating will be occurred in the controller.

The authors in [11][12] have proposed the idea of dynamic back-propagation (through the delay lines). The back-propagated delayed versions of the error can be appropriately added to get the error at the controller net output. Hence, we need to store past values of error at nodes to implement this back-propagation and this also requires some prior knowledge regarding the plant structure.

Another approach to deal with this issue is to use dynamic or recurrent neural networks [10][13][14]. These are trained directly as a parallel plant model but require more complex training algorithms, such as back-propagation through time [15], and hence are computationally intensive.

This is a brief reminder of neurocontrol issues, proposed schemes, and related limitations which have motivated our work. The basic idea outlined throughout the paper is to overcome the credit assignment problem with an identifier-free approach. The FEP learning algorithm developed in the next section is tailored especially for this purpose. It computes the output errors (*i.e.,* between the plant outputs and their desired values) at the controller inputs and then propagates them forward through the network to the get the hidden and the output layer errors required for the weight update.

Unified direct Learning control structure:

In the control literature, the *Indirect control* relies on a system identification procedure to form an explicit (or a parametric) model of the plant and determining the control rule from the model, whereas the *direct control* uses only an implicit model (or nonparametric identifier) to adjust the controller parameters. In neurocontrol, the identifier is used as a channel to back-propagate the error at the plant output up to the controller output. However, in the remain of this paper, we refer to *direct scheme* to the identifier-free design which does not need any form of explicit or implicit model of the plant(*Fig.1*).

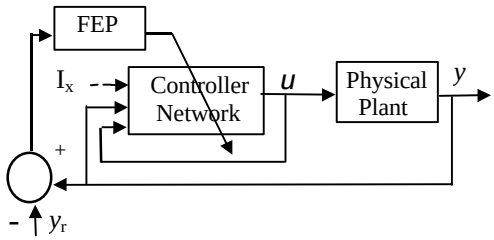

Fig.1 *FEP-based identifier-free learning control structure*

• If the control problem is expressed in terms of trajectory tracking (*i.e.,*STR [9], specialized inverse control [4] , or forward modeling [8]), then a neural network can be used to implement the control map $g_1(.)$ according to Fig.1. The desired trajectory can be fed through the external input I_x of the controller net. At each time step the error $e_c(k)$, between the plant output $y(k)$ and the desired trajectory $y_r(k)$, is fed by FEP though the network for the weight update.

• However in the MRAC systems, the control problem is expressed in terms of reference model following. The controller is trained to implement the map $g_2(.)$. It receives the reference input $r(k)$ and provides a modified value $u(k)$ so that the closed loop system results in the same mapping as the reference model. In this paper, we refer to this approach by '*parallel MRAC*' since the reference model is placed in parallel with the closed loop system (*Fig.2-(a)*).

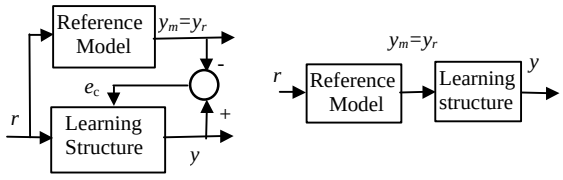

Fig.2. *Bloc diagram of the model reference adaptive control (MRAC)* ***(a).*** *Parallel structure* ***(b)*** *Series structure.*

Another way to solve the model following problem is to place the reference model in cascade with the learning structure of Fig.1. First, the reference model transforms the reference input $r(k)$ to the desired trajectory. Then, the learning structure is used for the trajectory tracking task. We refer to this approach by '*series MRAC'* since the reference model is in series with the closed loop system(*Fig.2-(b)*).

The main objective of this new structure is to provide a common scheme to train controllers. Furthermore, the parallel structure leads to a controller which is specific to the reference model used during training. If the latter is changed, the controller must be retrained. However in the series structure, the controller does not depend on the reference model. Even the latter is changed the same controller can be used.

• In optimal control, the controller net is trained, using a dynamic optimization method such as back-propagation through time [15], to implement a control map $g_3(.)$ which optimizes some performance criterion $J(k)$. No external input is required (*i.e.,* $I_x=0$). It only receives either the state vector or delayed values of the plant output and provides the control input $u(k)$ in optimal fashion.

If the performance index J is expressed only in terms of the feedback variables from the plant, then FEP-based training can be directly used. However, if J contains a term specifying the constraint on energy consumption(*i.e.,* $J(k)=J_y + J_u$) then a hybrid FEP/back-propagation training can be used. The part J_y is propagated forward using FEP, however the part J_u is back-propagated using the back-propagation algorithm. The total errors signals through the network are the sum of the two parts. If the controller has a feedback from its output then this can be used to feedforward propagate J_u through the controller network.

2. The FEP Training Algorithm

We know that a change in any synaptic weight in a neural network around its optimal (desired) value will affect the output of the final layer. The error measure on the network output J_N may be the result of wrong values of many synaptic weights. Therefore , the main purpose of a learning algorithm is to assign credit for each synaptic weight in the network. Back-propagation algorithm does this by propagating the output errors backwards through the network.

In control systems, as we have seen in the previous section, a sensitivity model is required to serve as a channel to back-propagate the error at the plant output up to controller output. However, we will show bellow that this can be done without of this model.

Since the controller takes as inputs the plant outputs, thus the change in these inputs around their desired values (desired state) will propagate through the controller network to produce a corresponding change in the controller output. This is the basic principle of FEP learning algorithm which recursively calculates the gradient matrix for each layer starting from the input layer and going forward layer by layer until the output layer is reached. This allows a direct fast computation of the hidden and output errors required for the weight update.

The algorithm derivation:

Let us consider the typical neural network with L layer with *0*th layer holds the input vector components (*i.e.,*$u_{0,j} = x_j$). The output of the *j*th node in any layer *l* ($0 < l \leq L$) is given by

$$u_{l,j} = f(S_{l,j}) = f\left(\sum_{m=0}^{N_{l-1}} w_{l,j,m} \cdot u_{l-1,m}\right) \qquad (2)$$

where $f(.)$ is the activation, $w_{l,j,i}$ is the weight which connects the ith node in layer $(l-1)$ to the jth node in layer l, and N_{l-1} is the number of nodes in layer $(l-1)$.
In the realm of control the input vector has the general form $X(k) = [Y_n(k), U_m(k), I_x(k)]$ with $Y_n(k)$ and $U_m(k)$ represent the feedback information flow, as they are defined in (1), and $I_x(k)$ stands for the external input (*i.e.*, $y_r(k)$ or $r(k)$). In FEP algorithm, the errors on plant outputs are directly injected via the feedback lines. Thus, all the components of error input vector, $E_0 = [e_{01} , e_{02} , \dots]$, are set to zero except the ones corresponding to the feedback from the plant outputs. Hence the error function to be minimized can be expressed by

$$J = \frac{1}{2}\sum_{q=1}^{n_y}(y_q - y_q^d)^2 = \frac{1}{2}\sum_{q=1}^{n_y} e_{0,q}^2 \qquad (3)$$

where y^d_q is the desired value of the qth output of the plant and n_y is the number of the plant outputs. The weights are updated iteratively according to

$$w_{l,j,i}^{new} = w_{l,j,i}^{old} - \mu \frac{\partial^+ J}{\partial w_{l,j,i}}|_{w^{old}} \qquad (4)$$

where μ is the learning rate. For an arbitrary weight in layer l, the ordered derivative $\partial^+ J_p / \partial w_{l,j,i}$ can be expressed by the chain rule as

$$\frac{\partial^+ J}{\partial w_{l,j,i}} = \frac{\partial^+ J}{\partial u_{l,j}} \frac{\partial u_{l,j}}{\partial w_{l,j,i}} = \frac{\partial^+ J}{\partial u_{l,j}} f'(S_{l,j}).u_{l-1,i} \qquad (5)$$

where $f'(s_{l,j}) = d f'(s_{l,j})/ds_{l,j}$ and the term $\partial^+ J/\partial u_{l,j}$ is the sensitivity of J to the output of node $u_{l,j}$. At the input layer we have a boundary condition where

$$\frac{\partial^+ J}{\partial u_{0,j}} = (u_{0,j} - u^d_{0,j}) = e_{0,j} \qquad (6)$$

For the internal node output $u_{l,j}$ $(0< l \le L)$, the partial derivative $\partial J(w)/\partial u_{l,j}$ is equal to zero , since J does not depend on $u_{l,j}$ directly. However , it is obvious that J does indirectly depend on $u_{l,j}$ since a change in $u_{l,j}$ around its desired value will propagate through both the controller and the plant and thus affects the plant output. Therefore, using this fact and by extending the meaning of (6) for the hidden layers, the hidden layer error signals $e_{l,j}$ can be though of as the change in $u_{l,j}$ around its desired value, $u^d_{l,j}$,

$$e_{l,j} = \Delta u_{l,j} = (u_{l,j} - u^d_{l,j}) \qquad (7)$$

Since each output $u_{l,j}$ is expressed in term of the node outputs of layer $(l-1)$ according to (2), hence the change $\Delta u_{l,j}$ can also be approximated by the changes $\Delta u_{l-1,m}$ $(m=1,...,N_{l-1})$ as follows

$$\Delta u_{l,j} = \sum_{m=1}^{N_{l-1}} \frac{\partial f(s_{l,j})}{\partial u_{l-1,m}} \Delta u_{l-1,m} \qquad (8)$$

Using (7), we rewrite (8) as

$$e_{l,j} = \sum_{m=1}^{N_{l-1}} \frac{\partial f(s_{l,j})}{\partial u_{l-1,m}} e_{l-1,m} \qquad (9)$$

This expression allows us to compute sequentially the errors at each layer as a linear combination of the errors of the previous layer.

At the beginning of the training process, the weights are initiated at random values and (9) represents only rough approximations of the error signals. As the learning takes place, the weights go to their right values. The error signal magnitudes decrease. Equation (9) becomes, mathematically, more and more rigorous (since it tends to the differential, $e_{l,j} = du_{l,j}$) allowing, hence, concise estimations of the errors signals. This qualitatively explain the best convergence property of the algorithm observed during the extensive simulation studies.

3. Simulation Studies

In this section, our interest is to demonstrate the efficiency of the proposed approach. The learning system of Fig.1 is quite general and can be applied to a variety of learning control problems. Here, three examples concerning the two classes of control problems are studies. The aim is to construct direct neural controllers for nonlinear dynamic plants which are assumed to be unknown, except some empirical assumptions concerning their behavior (*e.g.*, stability, controllability).

3.1 Direct Neural Self-Tuning Control

The basic idea is to train the controller network so that the composed closed loop system results in an identity mapping between the desired trajectory and the plant output. At each step k the controller receives, in addition to the feedback variables, the desired trajectory value $y_r(k)$ and provides to the plant the command signal $u(k)$. Then, the difference between the actual plant output $y(k)$ and $y_r(k)$ is propagated forwards through the controller, allowing a direct fast updating of its parameters without the use of

identifier. This makes FEP much suitable for « *on-line*» adaptive control. Because in adaptive systems it is assumed that the controller parameters are adjusted all the time according to changes in the plant. However in this paper, for the sake of simplicity, the plant is supposed to be time-invariant. In the presence of time-variant plants, the learning can be carried *on-line*.

The following example discusses the control of a nonlinear multivariable plant with two inputs and two outputs described by

$$\begin{bmatrix} y_1(k+1) \\ y_2(k+1) \end{bmatrix} = \begin{bmatrix} \dfrac{y_1(k)}{1+y_2^2(k)} \\ \dfrac{y_1(k)\,y_2(k)}{2(1+y_2^2(k))} \end{bmatrix} + \begin{bmatrix} u_1(k \\ u_2(k \end{bmatrix} \qquad (10)$$

The controller network, with the 15 neurons in the first hidden layer and 10 units in the second one, was trained during 10000 time steps using random inputs y_{r1}(k) and y_{r2}(k) uniformly distributed in the interval [-2,+2].

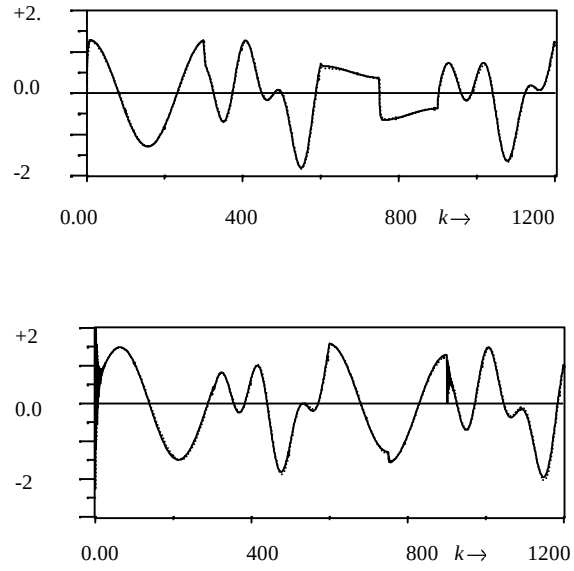

Fig.3 shows the outputs of the controlled plant and the desired trajectories. One can see the good tracking performance of the control system for different forms of the desired trajectories

3.2 Direct Series Neural MRAC

In this section, simulation results for the control of a nonlinear plant using the proposed series MRAC are reported. The plant is described by the following model

$$y(k+1) = \frac{y(k)}{1+y(k)^2} + u(k)^3 \qquad (11)$$

The reference model is described by

$$y_m(k+1) = 0.5y_m(k) + r(k) \qquad (12)$$

The aim is to choose u(k) so that $\lim_{k\to\infty} |y(k)-y_m(k)|=0$. The controller net, with the same structure as the one used above, was trained during 8500 time steps using a random input r(k) uniformly distributed in the interval [-1,+1].

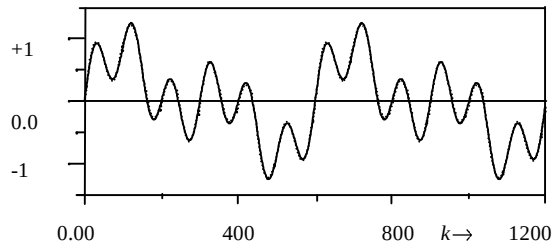

Fig.4 illustrates the FEP-based controller performance. One can see that the reference model and the plant outputs are almost the same. This shows the best generalization and convergence properties of FEP algorithm associated with less computation and mathematical representation complexities. This confers great promise for FEP in the realm of on-line neural adaptive control of time-variant nonlinear dynamical plants.

3.3 Direct Optimal Control of the Inverted Pendulum System

The dynamics of the inverted pendulum are characterized by 2 state variables θ(angle of the pole with respect to the vertical axis) and θ̇ (angular velocity of the pole). The control goal is to stabilize the pole in the vertical position by supplying an appropriate force to the cart. The neural controller, with 10 units in the first hidden layer and 5 units in the second one, was trained for 250 trials using self-

learning method [16]. At the end of each trial, the error on the state is propagated forwards through the controller and the network weights were updated according to FEP equations.

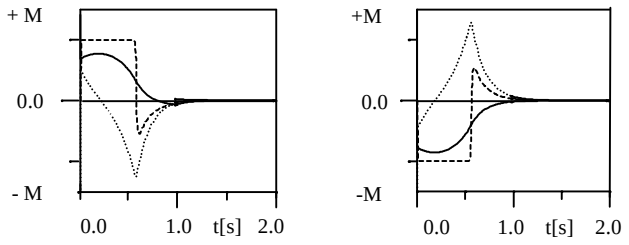

Fig.5. *Variations of the angle θ (—) , the angular velocity θ˙ (---), and the force F (— —) for the initial conditions(θ [°], θ˙ [°/s]):* ***(a)*** *(35,45)* ***(b)*** *(-35,-45). The* ***« M »*** *value corresponds to 12 N, 45°, and 100 °/s for F, θ, and θ˙ , respectively*

.

Fig.5 illustrate the obtained simulation results. One can see that the controller stabilizes the pole in the desired state during short periods of time. This can be explained by the efficiency of FEP in the error calculations and, in the other hand, by the forward model uncertainty of indirect self-learning.

4. Conclusion

With FEP algorithm, we have proposed a new approach for designing neural controllers. It provides credit assignment in a direct manner without the use of identifier or any form of the plant model. This greatly simplifies the learning complexity and avoids the forward model uncertainty and limitations. The simulation results show the best generalization and convergence properties of FEP algorithm associated with less computation and mathematical representation complexities. Although in this paper, we have shown only how to construct direct neurocontrol methods, FEP can be used to other kinds of learning systems.

References:

[1] K. S. Narendra and S. Mukhopadhyay,' *Adaptive Control Using Neural Networks and Approximate Models,'* IEEE Trans. On Neural Networks, vol.8, pp.475-485, May 1997.

[2] H. Ishibuchi, R Fujioka, and H. Takana,'*Neural Networks That Learn from Fuzzy If-Then Rules'* IEEE Tran. On Fuzzy Sys.,vol.1,pp.85-98,May 1993.

[3] R.Abu Zitar and M.H.Hassoun,'*Neurocontrollers Trained with Rules Extracted by a Genetic Assisted Reinforcement Learning System,'* IEEE Trans. On Neural Networks, vol.6, pp.859-879, July 1995.

[4] D.Psaltis, A.Sideris,and A.Yamamura,*"A mult-layered neural net controller"* IEEE contr. syst. Mag., vol.8,Apr. 1988.

[5] H.C.Andersen, F.C. Teng, and A. C. Tsoi, '*Single Net Indirect Learning Architecture,' '* IEEE Trans. On Neural Networks, vol.5,pp.1003-1005,Nov. 1994.

[6] P.J. Werbos, *"Consistency of HDP applied to a simple reinforcement learning problem,"* IEEE Trans. neural net., vol.3, pp.179-189, mar. 1990.

[7] R.S.Sutton, A.G.Barto, and R.J. Williams,*" Reinforcement leearning is direct adaptive optimal control,"* IEEE cont. sys. mag., vol.12, Apr. 1992.

[8] M.I.Jordan and D.E.Rumelhart, "*Forward models: Supervised learning with a distal teacher,"* Center for Cognitive Science, M.I.T., 1991.

[9] F.C. Chen, *"Back-propagation neural networks for nonlinear self-tuning adaptive control,"* IEEE control sys. mag., vol.10, pp.44-48, Apr. 1990.

[10] P.S.Sastry,G. Santharam, & K.P. Unnikrishnan, '*Memory Neuron Networks for Identification and Control of Dynamical Systems,'* IEEE Trans. On Neural Networks, vol.5, pp.306-319, March 1994.

[11] K.S. Narendra & K. Parthasarathy,*" Gradient methods for the optimisation of systems containing neural networks,"* IEEE Trans. on neural net.,vol.2, pp.252-262, mar. 1991.

[12] K. S. Narendra and K. Parthasarathy, *" Identification and control of dynamical systems using neural networks,"* IEEE Trans. neural net., vol.1, no.1, mar.1990

[13] G.V.Puskorius & L.A.Feldkamp,' *Neurocontrol of Nonlinear Dynamical Systems With Kalman Filter Trained Recurrent Networks,'* IEEE Tran. On Neural Networks, vol.5, pp.279-297, March 1994.

[14] M. K. Sundareshan and T. A. Condarcure,' *Recurrent Neural-Network Training by a Learning Automaton Approach for Trajectory Learning and Control System Design,'* IEEE Trans. On Neural Networks, vol.9, pp.354-368, May 1998.

[15] P.J.Werbos,*"Backpropagation through time: what it does and how to do it,"*proc.of IEEE,vol.78,Oct 1990

[16] D. H. Nguyen and B. Widrow ,*" Neural networks for self-learning control systems,"* IEEE cont. syst. mag., vol.10, No.3, pp.18-23, Apr. 1990