Applying Backpropagation through Time to a Real Inverted Pendulum Problem

M. Costa, F. Palma, D. Palmisano, E. Pasero
Politecnico di Torino, Dipartimento di Elettronica
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
e-mail: costa@polito.it, palmisano@mac2.polito.it, pasero@polito.it

Keywords: Neural networks, control systems, dynamic systems

ABSTRACT

The "inverted pendulum problem" is perhaps the most widely used benchmarking study to assess the effectiveness of emerging control design techniques. In this paper the "Backpropagation Through Time" learning method is used to train a multilayer perceptron neural network to control an actual physical system, consisting in a pendulum free to pivot on a cart, which is moved by a DC motor within a space of 60 cm on a pair of slide guides. The goal of the neural controller is to maintain the inverted pendulum balanced at the middle of the slide guides. The neural network has been implemented both with a DSP and with a Fast Prototipyng Neural System (FPNS) hardware. The experimental results show the effectiveness of the used technique: the network is able to balance the pendulum also from difficult initial conditions.

INTRODUCTION

The problem of the "inverted pendulum" is a classic topic in the field of automatic control, for it deals with an intrinsically unstable system, of relatively simple realization, whose dynamics is similar to those involved in tasks such as walking and the control of rocket thrusters [1], [2], [3]. Although a number of neural techniques has been used with simulated systems, little are those involving actual mechanical pendulums and hardware neural networks.
Classical control theory tries to linearize the pendulum behavior close to the equilibrium position (a few degrees). This approach is able to maintain this position also with external forces trying to perturbate the position. But out of the linearized interval the model looses precision and the system can become unstable or even the pendulum can fall down. The neural network doesn't try to linearize the problem and shows good performances also quite far from the equilibrium position.
In a first approach we used the same equipment used to control the pendulum by means of a classical P.I.D. We only replaced the control algorithm with our net. In a second approach we used our Fast Prototyping Neural System (FPNS) [4] to implement the neural net and to control the pendulum engine.
Next section describes the mechanical system whereas the following section describes the learning algorithm we used to add the dynamic behavior to a multilayer perceptron. Then we describe the cost function we studied to get the pendulum equilibrium. The training strategy is then described and the results obtained with the actual system are shown.

THE MECHANICAL SYSTEM

With reference to Figure 1 the system can be described as follows: a pole is pivoted on a moving cart, the cart being constrained through ball bearings on a pair of slide guides 60 cm long and moved by a force obtained with a DC motor. The motor's torque C_m gets transmitted to the cart through a timing belt and a couple of gears. The state of the system is determined by the following set of state

variables: α (angle between the vertical axis and the pole), $\dot{\alpha}$ (angular speed of the pole), x (cart position with reference to the middle of the slide guides), $\dot{x}$ (cart speed).

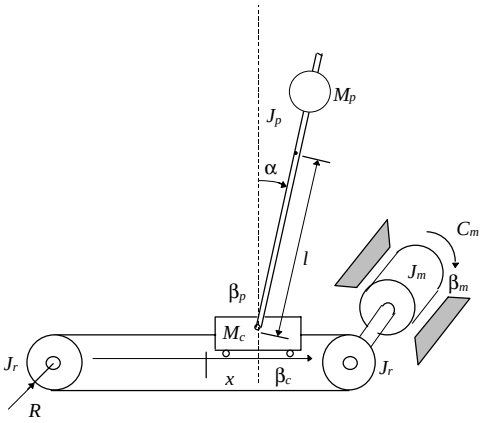

Figure 1: the mechanical system

The parameters characterizing the system are: J_m (moment of inertia of the motor's rotor), β_m (torsional viscous friction coefficient of the motor's rotor), J_r (moment of inertia of the gears), R (radius of the gears), M_c (mass of the cart), β_c (viscous friction coefficient between the slide guides and the cart), M_p (mass of the pendulum), l (distance between the pivot and the mass center of the pendulum), J_p (torsional viscous friction coefficient at the pivot); if C_m is the driving torque, the equations of motion are:

$$\ddot{x} = \frac{RC_m - (\beta_m + \beta_c R^2)\dot{x} - R^2 M_p (l\dot{\alpha}\cos(\alpha) - l\dot{\alpha}^2 \operatorname{sen}(\alpha))}{2J_R + J_m + M_c R^2 + M_p R^2} \qquad \ddot{\alpha} = \frac{-\beta_p \dot{\alpha} + M_p g l \operatorname{sen}(\alpha) - M_p l \ddot{x}\cos(\alpha)}{J_P}$$

(The dry frictions are not included in this model because the kind of learning algorithm we use requires, as discussed later, a differentiable description of the system to be controlled). These equations have been used to simulate the system with a fourth-order Runge Kutta algorithm.

LEARNING ALGORITHM

To control the inverted pendulum we make use of the "feedback control" scheme in which we suppose to have the measures of the four state variables. The neural controller is a multilayer perceptron with four inputs (the state vector), a single hidden layer and an output layer with only one unit. The output signal of the neural controller is sent to a power amplifier which drives the DC motor to produce the motor-torque C_m.

The aim of the learning process is to train the neural controller to maintain the inverted pendulum balanced at the middle of the slide guides, and the specific algorithm used employs the "Backpropagation Through Time" (BTT) learning method, that allows introduction of the temporal factor in neural networks. A description of BTT can yet be found in the work of Rumelhart et al. [5]; a more detailed treatment is in [6] and an application to a motion-control problem is in [7].

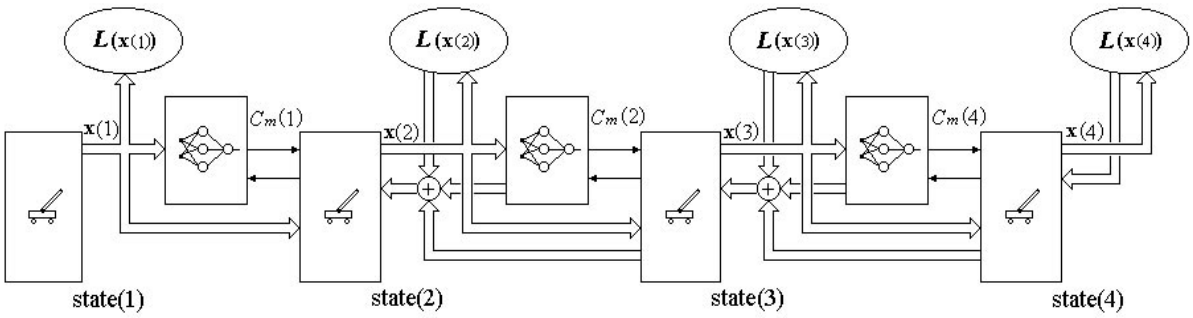

Figure 2: overview of a forward control sequence and of the subsequent BTT

The neural control operates with discrete time, so the state of the mechanical system is sampled every Δt seconds and passed to the neural controller. At its turn, the controller computes the control

signal in a negligible time, and such signal is held constant for the entire interval Δt. The network can reach the target of maintaining the inverted pendulum balanced at the middle of the slide guides if it is able to generate a sequence of commands that drive the cart and the pendulum from an initial position to the desired final state (all state variables equal to zero).
A general overview of a forward control sequence and of the subsequent Backpropagation Through Time is shown in Figure 2. This kind of learning is an atypical case of supervised learning for the only known target is the desired final state, state(f)=(0,0,0,0), while are not known both the proper controller outputs and the intermediate states of the mechanical system.
At this point two important questions arise. The first regards the availability of a neural emulator of the mechanical system, which is to be used during the backpropagation phase. The emulator can be obtained by linking a neural network in parallel to the mechanical system, so that even a traditional supervised learning can be employed to train the network to copy the system dynamics. This kind of procedure can also eliminate the traditional modeling and identification phases if the actual physical system is used for emulator training. Another possible way to get the emulator is to compile, after proper data acquisition process, a discrete look-up table where, given a time interval of interest, the final values taken on by the state variables are expressed in terms of the initial ones and of the action one wishes to perform. In our case however, since the actual mechanical system used for our experiment had been used in the past few years for several non-neural control studies, an accurate model was already available. Only for accessibility issues we resorted to such model to estimate the dynamical behavior of the system during the training phase (during the backpropagation phase, for each temporal step, the sensitivity of the output of the model with respect to the motor torque and the state variables have been approximated with the finite difference method).
The second important question we have to face with concerns the duration of the sequence of controller commands: it depends on the initial state of the mechanical system and it isn't known in advance. As we will see in the section on training, this is not a critical parameter at the beginning of the training, and a network with satisfactory performances can be easily obtained. Once such network is available, an estimate of the duration necessary for a given initial condition can be accomplished by simulation, and the training can be driven toward higher performances. Moreover we have to consider that the minimization of the cost function interests not only the final state of the sequence, but the whole sequence so that the duration of the succession can be overestimated. The choice of a suitable cost function will be discussed later. The complete scheme for a single step is shown in Figure 3.

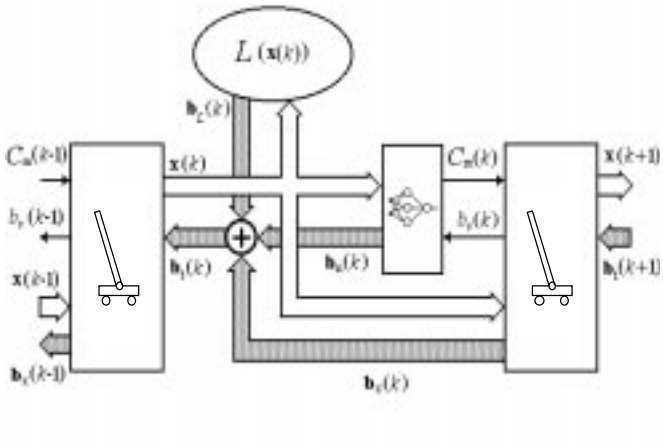

Figure 3: data flow

The right directed signals are relative to the normal control activity, the left directed ones represent the information relative to the derivative calculations that propagate backward in the system (and in time) during the BTT phase. The function ***L*** represented by the oval is a suitable cost function of the state $\mathbf{x}(k)$, whose derivatives $\mathbf{b}_L(k)$ have to be summed to these coming from the network, $\mathbf{b}_n(k)$,

and from the emulator, $\mathbf{b_x}(k)$ (at its turn the emulator receives the derivatives information from the temporal steps that in the control phase are subsequent); the resulting vector $\mathbf{b}_t(k)$ is then backpropagated through the emulator to the preceding time step, obtaining the signals $\mathbf{b_x}(k\text{-}1)$ and $b_c(k\text{-}1)$.

The cost function

The availability of a target point in the state space does not allow by its own direct application of a completely supervised approach to this kind of control problems. Actually a suitable differentiable cost function has to be defined that somehow measures the discrepancy between the current state and the desired one. The nature of the problem prevents from defining native cost function like the square Euclidean norm because contributions coming from the various state variables have to be homogeneous, while the variables themselves are not. Of course, individual contributions can be weighted, but then 4 more critical parameters have to be properly estimated mainly on a heuristic basis. We therefore pursued an alternative approach by directly attempting to minimize the lagrangian of the idealized pendulum, defined as:

$$L = E_c - E_p$$

where E_c is the kinetic energy and E_p the potential one. It is in fact apparent that the target state is a minimum for the kinetic energy and a maximum for the potential one.

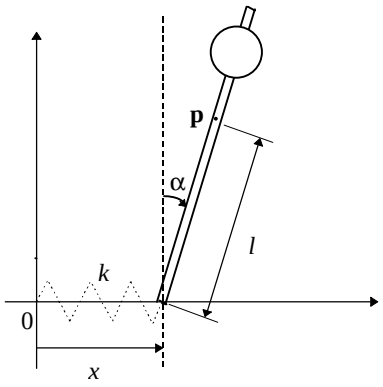

Figure 4 : the idealized pendulum with the virtual spring

0 : middle of the slide guides
x : distance between pivot and the middle of slide guides
p : centre of mass of the pendulum
l : distance between pivot and centre of mass
α : angle between the vertical axis and the pole
J_0 : moment of inertia of the pendulum relative to **p**
M_p: mass of the pendulum

With reference to the Figure 4 the kinetic energy is $E_c = \frac{1}{2} M_p \left(\dot{x}^2 + 2\dot{x}\dot{\alpha} l \cos(\alpha) + l^2 \dot{\alpha}^2 \right) + \frac{1}{2} J_0 \dot{\alpha}^2$;

the gravitational potential energy is $E_{pg} = M_p g l \cos(\alpha)$; however this expression does not include information about the distance between the middle of the slide guides and the pivot; this kind of information can be given by an additional contribution coming from a *virtual spring* shown with dashed line in Figure 4, that involve the potential energy:

$$E_{ps} = \frac{1}{2} k x^2$$

The expression for the L (that now is no longer a true lagrangian) is then: $L = E_c - E_{pg} + E_{ps}$

$$L = \frac{1}{2}\left(M_p\left(\dot{x}^2 + 2\dot{x}\dot{\alpha}l\cos(\alpha) + l^2\dot{\alpha}^2\right) + J_0\dot{\alpha}^2\right) - M_p gl\cos(\alpha) + \frac{1}{2}kx^2 .$$

The derivatives of $\boldsymbol{L}$ respect to the state variables, evaluated for $\mathbf{x}=\mathbf{x}(k)$, are the components of the vector $\mathbf{b}_L(k)$ in the Figure 3.
At this point is evident that the model of the mechanical system has to be differentiable; owing to this reason it is not possible to include the dry frictions in this kind of algorithm, since they cause remarkable discontinuity in the behavior of the system.

CONTROLLER TRAINING AND RESULTS

To train the neural-net controller we used the learning algorithm in a context of "lessons" of increasing difficulty, as hinted in [7]. At the beginning the cart and the pendulum are placed into initial condition very close to the target state (x in the range of ± 5 cm from the middle of the slide guides and α in the range of ± 5 deg), and the duration of the control sequence is fixed to 10 steps (these are non-critical parameters). When the controller is able to balance the pendulum, the duration of control sequence starting from specific initial conditions can be estimated through simulation, thus the training can be driven toward higher performances.
As example the initial learning curve of a network with 6 hidden units is shown in Figure 5; in this case only the final L is minimized, while the intermediates L are not considered (shows the L_m, i.e. the mean of the final L over the training set, consisting of 10 initial conditions).

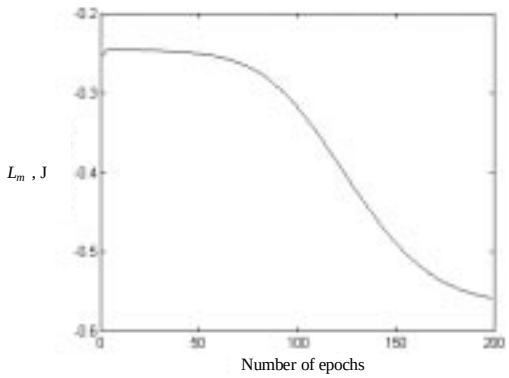

Figure 5: initial learning curve

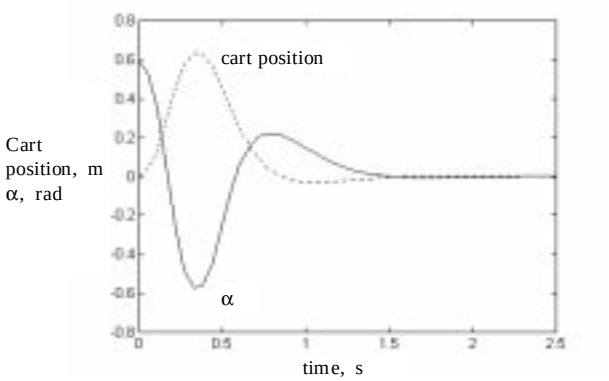

Figure 6: controller action

After 1500 epochs of learning, in which only the learning rate is modified, the controller shows good performances. To train the network until this stage take a few minutes with a 486DX2 66MHz processor. The performances of the controller can be improved by progressively extending the range of the initial conditions and estimating by simulation the needed duration of the sequence for the more difficult initial conditions. The network in the preceding example was able, after an advanced training phase, to balance the simulated (non linear) system from an initial state with α=40 deg. The Figure 6 shows the response of the system with an initial angle of 34 deg, and is virtually overshoot-free.
The neural-net controller has been implemented both with a DSP and with a Fast Prototyping Neural System (FPNS) hardware [4] for the application to the actual physical system. As often happen in control system, not all the state variables are available; in our case there are only 3 transducers on the system, and the angular speed $\dot{\alpha}$ is not measured, thus we did obtain it through an analogic differentiator. The results with the physical system (Figure 7 and Figure 8) confirm the good performances of the controller, though the presence of dry frictions gives rise to limit cycles.

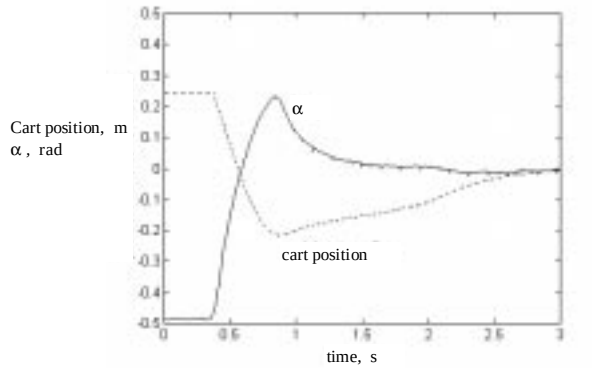

Figure 7: recover from α_i=27 deg, x_i= -25 cm

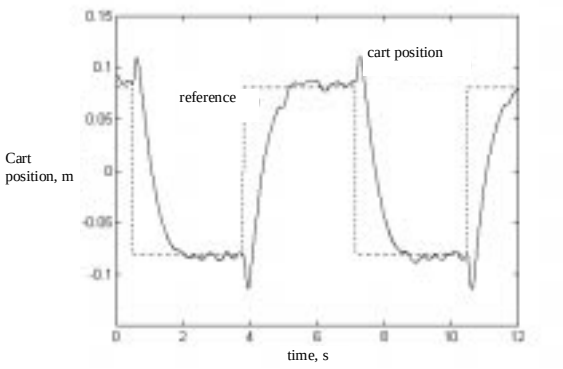

Figure 8: square wave reference

CONCLUSIONS

The paper describes a neural system able to control an inverted pendulum placed on a mechanical cart. Main topics of the work are the use of the time dimension in a Multilayer Perceptron, the definition of a particular cost function able to perform a data driven training of the neural network. The neural system demonstrated a good performances also where classical control systems loses rehability. A neural hardware controller was used to control the inverted pendulum.

REFERENCES

[1] A. G. Barto, R. S. Sutton, C.W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems*, IEEE Transactions on Systems, Man, and Cybernetics, SMC-13, pp. 834-846 (1983).

[2] M. I. Jordan, , D. E. Rumelhart, *Forward models: Supervised learning with a distal teacher*, MIT Center for Cognitive Science Occasional Paper #40 (1990).

[3] W.T. Miller III, R. S Sutton., and P. J. Werbos, eds, *Neural Networks for Control*, Cambridge, MA, MIT Press. (1990).

[4] G. Grossi, R. Posenato, M. Costa, D. Palmisano, E. Pasero, *Fast prototyping for hardware neural networks*, Proc. International Conference on Artificial Neural Network, vol.1. (1995).

[5] D. E. Rumelhart, G. E. Hinton, and R.J. Williams, "*Learning internal representations by error propagation*," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.), Vol. 1, Chapter 8, Cambridge, MA, MIT Press (1986).

[6] P. J. Werbos, *Backpropagation Through Time: What it does and how to do it*, Proceedings of the IEEE 78, pp. 1550-1560 (1990).

[7] D. Nguyen, B. Widrow, *The Truck backer-upper: An example of self learning in neural networks*, International Joint Conference on Neural Networks, vol 2, pp. 357-363, Washington, DC. (1989).