*Proceedings of the 1993 World Congress on Neural Networks, Portland, Oregon, Vol. 4, pp. 518-522, July 11-15, 1993.*

# Learning Control Tasks by Failure Signals

Daniele Micci Barreca and Giorgio C. Buttazzo

A.R.T.S. Lab[1] - Scuola Superiore S. Anna
Via Carducci, 40 - 56100 Pisa, Italy

## Abstract

In this paper we present a hybrid neural architecture for learning motor control actions by failure signals. Three neural paradigms have been integrated in the architecture: competitive learning is used for coding status information; reinforcement learning is employed to drive control actions according to an evaluation function; and supervised learning is used to construct a secondary reinforcement signal needed to handle the problem of temporal credit assignment. As a test bed for verifying our neural architecture we have considered the well-known cart-pole control problem. Experimental results obtained from simulations showed that better performance can be achieved with respect to other proposed architectures, in terms of both learning speed and generalization.

## 1. Introduction

Failure-based learning is a particular reinforcement learning paradigm, in which the only source of information for modifying the synaptic weights of neurons derives from a sequence of negative failure events, experienced by the system [1][2]. According to this learning paradigm, the neural network learns under the influence of some sort of evaluative feedback, which generally assesses the performance of the entire network, rather than the performance of individual elements.

As proposed by Anderson [1], the process of learning to control a dynamic system, under a failure-based paradigm, involves the acquisition of two functions: an *action function*, which associates control actions to system states, and an *evaluation function*, which assigns an evaluation to each possible state. Through the evaluation function credit, or blame, can be assigned to the individual actions produced by the learning system.

When such functions are non linear, a single-layer neural network cannot form this map. One solution to this problem is to transform the original state variables into a new representation with which a single-layer network can form the evaluation function. Barto et al. [2] demonstrated that a quantization of the state space of the system allows the use of single-layer networks to map a non linear evaluation function, as well as a non linear action function. A second solution is to make use of multi-layer networks to learn such a representation. Anderson [1] applied the Back-Propagation method to a two-layered network for solving the pole-balancing problem.

Previous efforts in the area of neural control are too numerous to exhaustively list here [7][11]. We instead list some of the works related to our approach, that have inspired us to develop the proposed method. Barto [3] presented a broad overview of neural architectures for controlling dynamic systems. Gullapalli [5] proposed a reinforcement paradigm for learning real-value functions, and applied it to perform a peg-in-hole insertion task [6]. Buttazzo et al. [4] used a similar method for tracking moving objects with a retina-like visual sensor rotating in two-degrees-of-freedom. Other approaches on reinforcement learning for control are presented in [13][16].

In this paper we propose a hybrid neural architecture for controlling dynamic systems under a failure-based learning paradigm. Three neural paradigms have been integrated in the architecture: competitive learning is used for coding status information; reinforcement learning is employed to drive control actions according to an evaluation function; and supervised learning is used to construct a secondary reinforcement signal needed to handle the problem of temporal credit assignment. The performance of the proposed hybrid neural system has been evaluated by considering the problem of maintaining the equilibrium of a pole hinged on a moving cart, by applying forces to the cart body. This approach extends the work of Barto [2] by replacing the decoder with a self-organizing neural network, which provides an adaptive state variable quantization based on the actual states experienced by the system.

---

[1] Advanced Robotics Technology and Systems Laboratory

## 2. The Hybrid Neural Architecture

The overall structure of the neural controller is shown in figure 1. The system comprises three main units: a Neural Decoder (ND), a Secondary Reinforcement Unit (SRU), and a Control Action Unit (CAU).

The ND is essentially a Kohonen self-organizing network [8], based on a *competitive learning* paradigm. A population of $N$ processing units receives the same $k$-dimensional input pattern, the system state vector, and classifies each pattern as belonging to one of $N$ non overlapping *regions* in which the input space is adaptively decomposed.

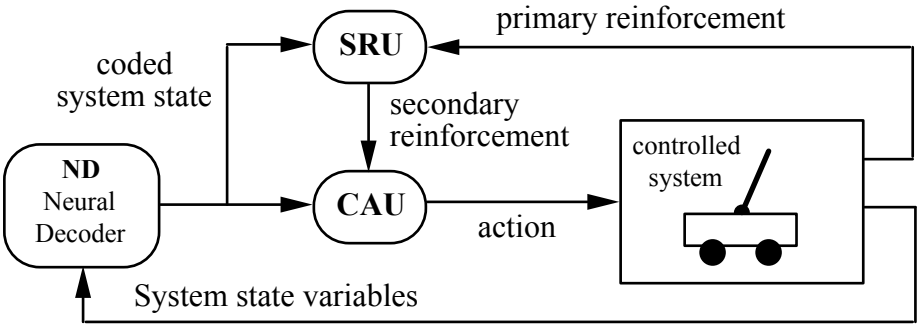


Figure 1: Block diagram of the neural architecture.

The main property of a self-organizing Kohonen's map is that it realizes a *data driven* quantizer, allowing to obtain a finer quantization over the most frequent states and a coarse one over the less frequent states. In this sense the quantization will always be *adequate* to the specific control task. Moreover, this adaptive process does not require any external supervision.

If compared with those neural systems where supervised methods are combined with reinforcement learning, like in Anderson [1], Gullapalli[5] and Lin [9], the ND can be related with the hidden layer of Back-Propagation units. In fact, the role of the hidden layer in such neural architectures is to provide a non linear transformation of the input patterns, so that the mapping performed by the output layer becomes linearly separable.

The Secondary Reinforcement Unit, SRU, is a perceptron-like unit trained to provide a *better* reinforcement signal when the external evaluation signal, supplied by the environment, occurs infrequently. In these cases the learning system has to solve the so called *temporal credit assignment problem* [1][12]. This is the problem of assigning the correct merit, or blame, to each action executed by the learning system in case of delayed evaluation. When a more informative grading signal is already available from the environment, this part of the system is no longer needed. Since our system has been specifically designed to work under a *failure-based* paradigm, the use of this unit is of crucial importance.

In our system, the SRU uses the method of Temporal Differences, called the TD(λ) algorithm [12]. to provide the CAU unit with an estimate of the expected primary reinforcement. The SRU receives the coded state pattern from the ND, and the *primary*, reinforcement signal from the failure sensors, and it produces a reinforcement prediction, anticipating punishment events.

The CAU is a stochastic binary valued unit responsible for the actual control actions of the learning system. Although the output of this unit is non deterministic, its statistical behaviour is biased through learning in order to achieve the maximum reinforcement over time. If a certain action is followed by reward, the weights are updated to make the action *more* probable in that given context. Symmetrically a punishment reduces the probability of the action being selected in that context. This algorithm was originally proposed by Barto [2] for the ASE/ACE control system.

## 3. The Learning Paradigms

Since different learning paradigms are involved in the system, each module is trained through a different learning law. The ND is composed of $N$ processing elements, having weights $\boldsymbol{w}_i \in \Re^k$ ($i = 1,2 \ldots N$), where $k$ is the dimension of the input vector and $N$ is the number of units in the network. A two-dimensional spatial topology is defined over the network, where the *topological neighbourhood* $N_i(R)$ for a unit $i$, is defined as the set of units within a radius $R$ around the unit $i$. For each learning step, the input vector $\mathbf{x} \in \Re^k$ is compared with the set of weight vectors, and a matching value is computed for each unit. The metric used in our system is the

Euclidean distance. The best-matching unit is declared the *winner* and the weights are updated for all the units in $N_{winner}$ through the following Kohonen's law:

$$\mathbf{w_i}(t+1) = \mathbf{w_i}(t) + k(i)\,\alpha(t)\,[\mathbf{x} - \mathbf{w_i}(t)] \qquad \text{if } i \in N_{winner}$$
$$\mathbf{w_i}(t+1) = \mathbf{w_i}(t) \qquad \text{otherwise}$$

The factor *k(i)* is called *neighbourhood interaction function*, and it is a decreasing function of the topological distance. Lo and Bavarian [10] have shown how the use of the neighbourhood interaction function enforces the topological ordering of the network.

The learning rate $\alpha(t)$ is a scalar parameter whose value is a decreasing function of the time. Also the size of the neighbourhood, i.e. its radius *R*, decreases with time. Usually, the initial value of the radius is set to cover the entire network, while the final value is set to zero.

The SRU builds a reinforcement prediction by using the TD(λ) algorithm. The complete explanation of the TD(λ) is out of the scope of this paper and can be found in [12]. In brief, the unit is trained using a supervised method, where the error at each step is proportional to the difference between the prediction at time *t* and the prediction at time *t+1*. In this way, the SRU is trained to predict its own predictions in order to anticipate the external reinforcements. The learning rule is the following:

$$P(t) = \mathbf{v}(t)^T \mathbf{c}(t)$$
$$\Delta\mathbf{v}(t+1) = \beta\,[P(t+1) - P(t)]\,\mathbf{Tr}(t) \qquad \text{where } \mathbf{Tr}(t+1) = \mathbf{c}(t+1) + \lambda\,\mathbf{Tr}(t)$$

where P(t) is the prediction at time t of the primary reinforcement, **v**(t) the weight vector associated with the input **c**(t) of the SRU.

When an external reinforcement signal arrives to the SRU, the prediction *P(t+1)* is substituted by the actual value of the primary reinforcement. The vector ***Tr(t)*** is called *activation trace*, and accounts for recent activations of each input path. The parameter λ controls the decay speed of the activation trace. A fast decay accounts for short *look-ahead*, since the communication of an external reinforcement will affect only those input paths having a significant activation trace.

The CAU is a non-linear adaptive element whose training is based on a *graded* random search of the input/output associations. A weight vector ***s*** is associated with the coded state vector ***c*** provided by the ND. Formally, the net input to the CAU is given by the inner product $\boldsymbol{s}^T\boldsymbol{c}$. A zero-mean gaussian noise η(t) is then added to the net input, and this sum determines the actual output *y* of the CAU by the following rule:

$$\text{if } \mathbf{s}(t)^T \mathbf{c}(t) + \eta(t) > 0 \qquad y = +1$$
$$\text{otherwise} \qquad y = -1$$

The learning rule updates the weight vector **s** to increase the probability of selecting rewarded actions and to decrease the probability of selecting punished actions. As in the SRU, the CAU keeps track of recently activated input paths so that delayed rewards and punishments can influence past actions. For each input path an *eligibility* value keeps track of the recent input/output correlation for that path. The eligibility is updated at each time step by the following rule:

$$e_i(t+1) = \delta\, e_i(t) + (1 - \delta)\, c_i(t)\, y_i(t)$$

where δ is the decay speed for the eligibility. The sign of the eligibility depends on the output of the unit, and this allows to modify weights in the correct direction by the following learning rule:

$$s_i(t+1) = s_i(t) + \rho\, r(t)\, e_i(t) \qquad 0 < \rho < 1$$

where $r(t) = P(t+1) - P(t)$, and ρ is the learning rate for the CAU. The secondary reinforcement *r(t)* is computed as the difference between the expected reinforcement at time *t+1* and the expected reinforcement at time *t*. Thus, when the new state has a higher predicted reinforcement than the previous one, the system is rewarded; whereas, when the predicted reinforcement is lower, meaning that the new state is *worse* than the previous one, the system is punished.

## 4. Simulation Results

To verify the behavior of the proposed neural control architecture, and to compare its performance with other related works, the system has been used to solve the classical *cart-pole problem*, also known as the *pole balancer* [1][2][14][15]. A cart is free to move back and forth on a bounded track, while a rigid pole hinged on

the top of the cart can rotate about a pivot. Both the cart and the pole are constrained to move in one dimension only. This learning control task requires the controller to keep the rigid pole about the vertical position by applying horizontal forces at the center of the cart. Since the output of the CAU is binary, the force has a constant module, while the versus (left or right) is determined by the neural controller. The state of the cart-pole system is described by means of four variables: the position and the speed of the cart, and the leaning angle and the angular speed of the pole. These variable are given as input to the neural controller.

A *failure event* is notified to the system by means of a number of "pain" sensors, which signal a failure when the pole angle exceeds a threshold value of 20 degrees, or the cart hits one of the two track bounds. These failure events are used as *primary* reinforcement signal for the SRU module.

The cart-pole system has been simulated by a dynamic model (without friction) as in [11]. The neural decoder ND has been organized as a 16x16 square matrix. The training of the system was conducted in the following way. At the beginning of every *trial*, which is a control sequence ending with a failure, the pole-cart system was positioned on a random location on the track, with the pole in the vertical position. On each occurrence of a failure, the primary reinforcement was set to $-1$, and the system was prepared for a new trial.

For each experiment, the pseudo-random number generator was initialized with a different seed, and the training lasted for 500,000 time iterations, corresponding to 2.7 hours of simulated time. The decay speed for the learning parameter $\alpha$ of the ND was set in order to make the Kohonen network stable after about 50,000 iterations.

Notice that an increase of performance will cause the system to experience failures less frequently, slowing down the overall learning process. For example, the system might quickly learn to maintain the equilibrium when the cart-pole is initially set on a particular position of the track, and hence cycling for long time within a small set of states for which the correct control law has been learned. However, since the system learns only through failures, no learning would occur in this phase, and even though the system is well performing around a specific state, it might be unable to work about another point far from that state. Since we are interested in a good general behavior, we want the system to experience failures as much as possible. For this reason very long, but rare, control sequences should be avoided. To overcome this problem we decided to interrupt a trial whenever its duration overcame a given number of time steps.

The plots in Figure 2 show the trial length for two different experiments, in terms of time iterations, versus the total training time; the dotted vertical line shows the end of the ND learning phase. Here the trials where forced to end after 50,000 iterations (about 16 minutes of simulated time). By comparing these two charts, the "negative" effect of early long sequences becomes evident. In the experiment 2, long sequences where reached much earlier with respect to the experiment 1, but these initial very good performance has been quickly followed by several failures. On the other hand, the system increased its performance more regularly in the experiment 1, where the system experienced more failures at the beginning of the training.

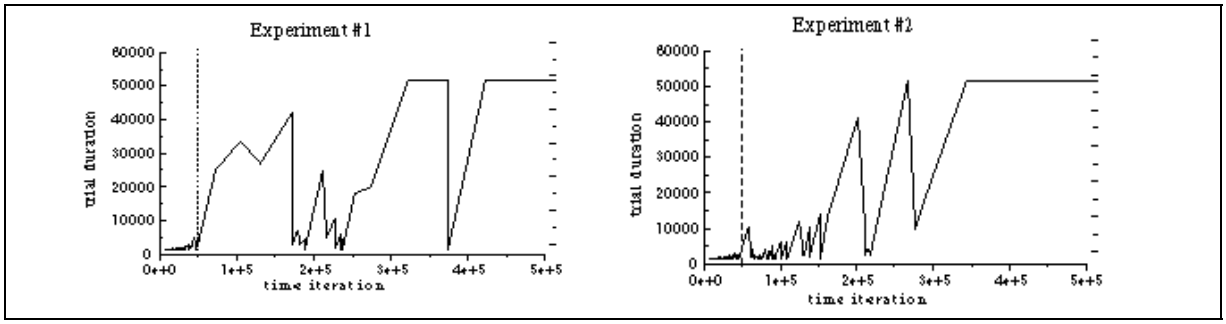


Figure 2: Time until failure vs. learning time, with a trial limit of 50,000 steps.

A second group of experiments were run with a length limit of 10,000 iterations. The benefits on the overall performance are evident from the charts shown in Figure 3. After about 150,000 iterations the system was able to balance the pole for the maximum allowed time at almost every trial.

Another relevant measure of performance is given by the number of failures required by the learning system to reach significant control abilities. Experiments have shown that after about 700 failures the system was capable of maintaining the equilibrium for quite long periods. This result can be directly compared with that of similar neural control systems, like the ASE/ACE system [2] or Anderson's multilayered network [1]. In the ASE/ACE system, the training phase was slightly different, in that the cart-pole was positioned at the centre of the track at the beginning of every trial. This clearly reduced the learning complexity, since less generalization was required. As a matter of fact, the system was able to reach a good performance after about 60 failures only.

In Anderson's experiments, where the cart was positioned in a random initial location, the system required more than 5,000 failures for learning.

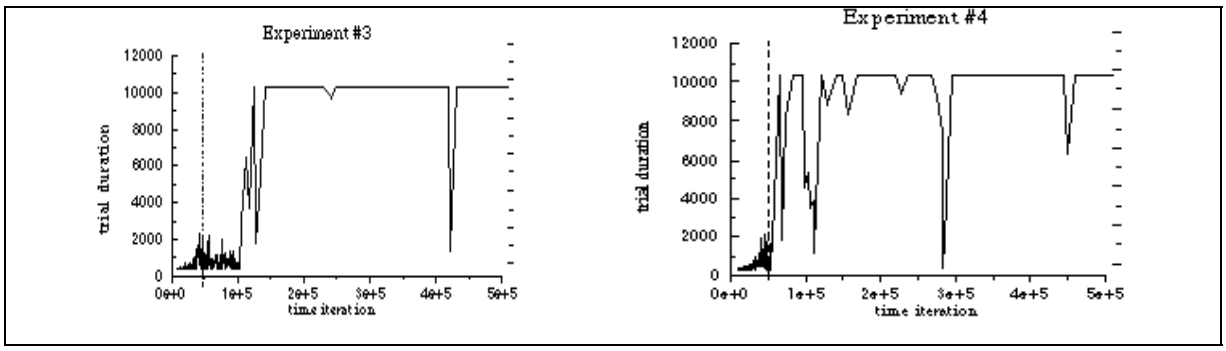


Figure 3: Time until failure vs. learning time, with a trial limit of 10,000 steps.

## 6. Conclusions

In this paper we have addressed the problem of learning a control action only based on a sequence of negative failure events experienced by the system. When a control objective cannot be expressed as an analytical function, failure signals are used for modifying the synaptic weights of neurons in the controller.

The neural architecture we proposed to support failure-based learning integrates three neural models, which allow to take advantage of different techniques for achieving generalization, unsupervised control, and dealing with the problem of temporal credit assignment, which may derive from the delay existing between actions and failures. To test the performance of the proposed neural architecture we have elected to consider the classical pole-balancing problem, in which a pole hinged from its base on a moving cart has to be balanced by applying forces to the cart. Experimental results obtained from simulations demonstrated a better performance in terms of both learning steps and generalization, with respect to other methods proposed in the literature for controlling the same system.

The significant improvements obtained by the proposed architecture, with respect to other hybrid systems using multi-layer networks, demonstrated that the use of a self-organizing map within a reinforcement learning neural system, represents a valid alternative solution to the problem of controlling dynamic systems, and encourage further investigation on this issue.

## 7. References

[1] C. W. Anderson: "Learning to Control an Inverted Pendulum Using Neural Networks", IEEE Control Systems Magazine, pp. 31-36, April 1989.

[2] A.Barto, R.Sutton, W.Anderson: "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems", IEEE Trans. on Sys., Man and Cybernetics, Vol. 13, pp. 834-846, Sept.-Oct. 1983.

[3] A.Barto: "Connectionist Learning for Control" in "Neural Networks for Control", Miller, Sutton, Werbos, eds., MIT Press, 1990.

[4] Buttazzo, G. C., P. Rege Cambrin, and L. Nardinocchi: "Unsupervised Back-Propagation for Visual Tracking", Technical Report A.R.T.S. Lab 92-03, Scuola Superiore S. Anna, Pisa, Feb. 1992.

[5] V. Gullapalli: " A stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions", Neural Networks, Vol. 3, pp. 671-692, 1990.

[6] Gullapalli, V., R. A. Grupen, and A. G. Barto: "Learning Reactive Admittance Control", Proc. of IEEE Int. Conf. on Robotics and Autom., Nice, France, pp. 1475-1480, May 1992.

[7] Horne, B., M. Jamshidi, and N. Vadiee: "Neural Networks in Robotics: A Survey", J. of Intelligent and Robotic Systems, No. 3, pp. 67-72, 1990.

[8] T. Kohonen: "Self-Organization ans Associative Memory", Springer-Verlag, 1984.

[9] L. J. Lin: "Self-improving Reactive Agents: Case Studies of Reinforcement Learning Frameworks", Technical Report, School of Computer Science, Carnegie Mellon University, 1990.

[10] Z. P. Lo, and B. Bavarian: "On the rate of convergence in topology preserving neural networks", Biological Cybernetics, Vol. 65, pp. 55-63, 1991.

[11] Miller, W. T., III, R. S. Sutton, and P. J. Werbos, Eds, Neural Networks for Control, MIT Press, 1990.

[12] R. Sutton: "Learning to Predict by the Methods of Temporal Differences", Machine Learning, No. 3, pp. 9-44, 1988.

[13] R. Sutton, A. G. Barto, and R. J. Williams: "Reinforcement Learning is Direct Adaptive Optimal Control", Proc. of the 1991 American Control Conference, 1991.

[14] V. V. Tolat, and B. Widrow: "An Adaptive 'Broom Balancer' with Visual Inputs", Proc. IEEE Int. Conf. on Neural Networks, San Diego, CA, pp. II 641-647, July 1988.

[15] B. Widrow: "The Original Adaptive Neural Net Broom-Balancer", Int. Symp. Circuits and Systems, pp. 351-357, May 1987.

[16] Williams, R. J., "On the Use of Back-Propagation in Associative Reinforcement Learning", Proc. of the IEEE Int. Conf. on Neural Networks, San Diego, 1988.