A KNOWLEDGE-BASE GENERATING FUZZY-NEURAL CONTROLLER

Ivan Petrović, Kristijan Maček, Nedjeljko Perić

Department of Control and Computer Engineering in Automation
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia

Abstract: - This paper presents a modification to the Kandadai and Tien's learning algorithm for tuning a fuzzy-neural controller that is able to automatically generate a knowledge base. Tuning is based on reinforcements from a dynamical system, thus giving a pseudosupervised learning scheme using error backpropagation. Originally, a weak reinforcement in the form of a binary failure signal was assumed which proved to be insufficient in terms of steady-state error. Therefore, a continuous reinforcement signal is applied enabling the system to correct the error as well as decreasing the overall control effort in the learning phase.

Keywords: expert systems, intelligent control, fuzzy logic, neural networks.

1. INTRODUCTION

Conventional controllers managing complex plants require some means of analytical modeling. However, in many practical systems the unavailability of quantitative data in terms of input/output relations makes knowledge-based controllers an interesting alternative. In this case, the analytical models are supplemented with a set of linguistic rules emulating the performance of a skilled human operator. Furthermore, such a fuzzy inference system can be integrated with neural networks, giving an adaptive structure with learning capabilities.

Thus, a possible architecture is the generalized approximate reasoning-based intelligent controller (GARIC) proposed by Berenji and Khedkar (1992), permitting the transformation of the network node weights into a linguistic rule base. Furthermore, Kandadai and Tien (1997) were able to modify the structure using Lin and Lee's approach (1996) to obtain a controller which would automatically generate its knowledge-base.

The problem statement assumed that the training data are very rough and coarse in form of a reinforcement signal, which is a scalar. This implied using a reinforcement learning scheme in contrast to supervised learning, where training data are available at each time step. Given that the reinforcement signal may be available at a time long after a sequence of actions has occurred, prediction capabilities were developed applying Sutton's temporal difference methods (1988). In general, multi-step prediction of reinforcement signal was required.

According to Kandadai and Tien (1997), reinforcement was a binary signal indicating whether or not the system has reached the failure state, where the controlled process was the cart-pole system. Thereafter, a hierarchical controller architecture was developed consisting of subcontrollers servicing specific subtasks. It was shown that the controller architecture was able to generate an extractable knowledge-base.

However, when applying the learning strategy to one subcontroller only, it was found that the proposed binary failure signal was not informative enough to compensate for the steady-state error. In this paper a continuous reinforcement signal is introduced leading to the single-step prediction case. Modifications of Kandadai and Tien's learning algorithm (1997) are made to ensure convergence and robustness. Finally, an application of the proposed architecture to the cart-pole balancing system is presented, taking in consideration some real-world problems such as friction in the bearings and measurement noise.

2. CONTROLLER ARCHITECTURE

The controller architecture proposed by Kandadai and Tien (1997) was based on the GARIC architecture. The main principles were elaborated on

a structure assuming that the plant model was known. However, it was subsequently shown that the need for a plant model could be eliminated. Whereas in the former case the structure was applied to a hierarchical controller, in the latter case only one subcontroller is considered. The controller consists of four main units: action evaluation network (AEN), action selection network (ASN), action search unit (ASU) and decision making unit (DMU) as shown in Fig. 1.

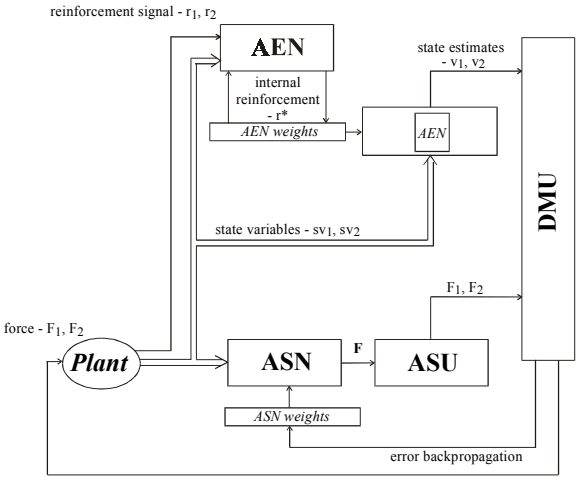

Fig. 1. A knowledge-base generating fuzzy-neural controller.

2.1 Action Evaluation Network (AEN)

This neural network is the same as in the GARIC architecture and is reproduced here. The AEN network receives the state variable values *sv* and the reinforcement signal *r* which it uses to generate an evaluation *v* of the system state. The approximate interval of values of *v* is [-1,0], where the value of -1 represents the failure state, while 0 represents the reference state. The structure of the AEN network is shown in Fig. 2 where **A**, **B**, **C** represent the weight banks of the neural network (Berenji and Khedkar, 1992).

The state score *v* is combined with the reinforcement signal *r* to give the internal reinforcement r^*. Based on Sutton's temporal difference methods (1988), the expression given in (Berenji and Khedkar, 1992) is as follows:

$$r^*[t] = r[t] + \gamma v[t] - v[t-1] \qquad 0 \le \gamma < 1 \qquad (1)$$

where t-1 and t are successive iterations and γ is the discount rate. Since here it is assumed that the reinforcement signal *r* is available at each iteration, multi-step prediction in (1) degrades to single-step prediction of the reinforcement signal:

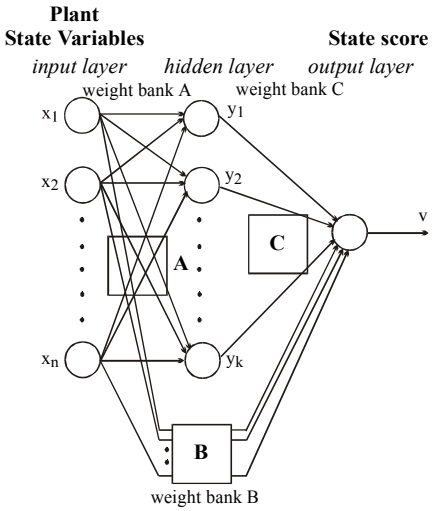

Fig. 2. Action evaluation network (AEN).

$$r^*[t] = r[t] - v[t-1]. \qquad (2)$$

Internal reinforcement r^* actually represents the prediction error of the AEN with respect to the reinforcement signal *r*.

2.2 Action Selection Network (ASN)

This structure enables the fuzzy inference scheme to be incorporated in a neural network, thus offering the advantage of a straightforward adaptation. Here the ASN network is a variation of the GARIC Action Selection Network (Berenji and Khedkar, 1992) in which all possible fuzzy rules are implemented (Kandadai and Tien, 1997). The structure is presented in Fig. 3.

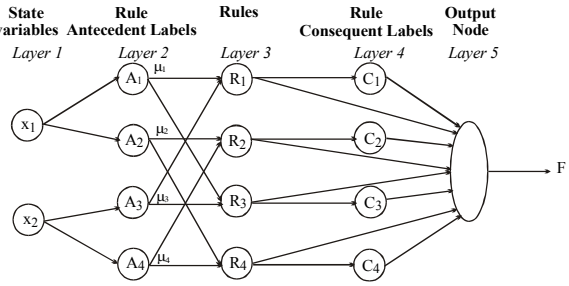

Fig. 3. Action selection network (ASN).

Since the ASN determines the output action values, the aim is to adjust the link weights of the network so as to maximize the expectation of the reinforcement signal *r*, which gives an evaluation of past actions chosen. The gradient information $\frac{\partial r}{\partial F}$, which is required if the pseudosupervised learning scheme is to be applied to the ASN, may only be estimated because the reinforcement signal *r* depends heavily

on the dynamics of the plant as well as on the previous actions chosen. Furthermore, it was shown by Kandadai and Tien (1997) that only weights in rule consequent labels need to be modified.

2.3 Action Search Unit (ASU)

Given the current state of the plant, the ASN suggests an output action F which is considered to be the expected value of some ideal action F'' which, if applied to the plant, would result in moving to an ideal next state. Since action F'' is not known to the learning system, there must be some uncertainty introduced in choosing the appropriate output action. This is achieved through stochastic exploration. A possible approach was analyzed in (Gullapalli, 1990) using multiparameter distributions. Based on this idea, Berenji and Khedkar (1992) and Kandadai and Tien (1997) determined the magnitude of deviation σ of the actual action F' applied to the plant with respect to the suggested value F by using an exponential function with internal reinforcement r^* as an indicator of state score improvement. In this paper, the difference between two successive state scores v is used, leading to the following relation:

$$\sigma[t] = \alpha_1 \exp(-\alpha_2 \Delta v[t]) \tag{3}$$

where $\Delta v[t] = v[t] - v[t-1]$ and α_1, α_2 are positive constants. If the system exhibited an improvement in the plant state score (i.e. positive increase) over the previous time step, then the uncertainty in choosing the output action F' should be smaller at the current time step or vice versa in the case of aggravation of the plant state score.

2.4 Decision Making Unit (DMU)

Given the present state of the plant, this unit determines the series of actions that should be applied to the plant for it to follow an ideal trajectory of the state variables based on the maximum state improvement. The selection of the actual action F' applied to the plant in relation to the suggested value F depends on the learning strategy for the ASN, as described in Subsection 3.2.

3. LEARNING MECHANISMS

3.1 Learning in AEN

As mentioned in Subsection 2.1, the internal reinforcement r^* represents the measure error of the AEN, which is used to modify the weight coefficients a_i, b_i, c_i, of weight banks **A**, **B**, **C** (Fig. 2) according to the error backpropagation algorithm (Rumelhart, *et. al.*, 1986), which is modified as outlined in (Barto, *et. al.*, 1983).

3.2 Learning in ASN

In order to extract the learning scheme for adjusting the weights of the ASN when the plant model was not known, the following assumptions were made in (Kandadai and Tien, 1997):

1) The ideal action F'' is assumed to be greater than or lesser than the action F suggested by the ASN.
2) The time step is sufficiently small, so that the plant states at two successive iterations are close enough to cause the ASN to suggest approximately the same expected action F.

According to Assumption 1, the ASU estimates the ideal action F'' by stochastically generating two actions F^+ and F^- each of which is greater than and lesser than the ideal action F''. Assumption 2 implies that at any two successive iterations a and b the ASN gives approximately equal expected actions $F_a \approx F_b \approx F$. The ASU then uses F_a to generate F^+ and, at the next iteration, it uses F_b to generate F^-. Therefore, the DMU receives F^+ at iteration a, and F^- at iteration b, and so on. It applies whichever action F^+ or F^- it receives from the ASU. When F^+ is applied to the plant at iteration a, the plant moves to a state with a state score v^+ at iteration b. Thereafter, when F^- is applied at iteration b, the plant moves to a state with a state score v^- at the next iteration a. The DMU then has to decide which action F^+ or F^- is closer to F''. Since it is possible that both F^+ and F^- can cause an improvement in the state score, the DMU chooses the action which resulted in the greatest improvement. The learning rule to adjust the parameters p of the ASN is then according to (Kandadai and Tien, 1997):

$$\Delta p = \eta \Delta F \Delta(\Delta v) \frac{\partial F}{\partial p} \tag{4}$$

where η is the learning rate, $\frac{\partial F}{\partial p}$ the sensitivity function and $\Delta F \Delta(\Delta v)$ a heuristic learning signal:

$$\begin{aligned} \Delta F \Delta(\Delta v) &= (F_1^+ - F_2^-)(\Delta v^+ - \Delta v^-) = \\ &= (F_1^+ - F_2^-)\left[(v_2^+ - v_1^-) - (v_3^- - v_2^+)\right] = \\ &= (F_1^+ - F_2^-)\left[2v_2^+ - v_1^- - v_3^-)\right] \end{aligned} \tag{5}$$

where indices 1,2,3 represent three successive time steps.

The proposed learning scheme was applied to the multistep-prediction case described in (1) where the reinforcement signal *r* was of the form:

$$r[t] = \begin{Bmatrix} -1 & failure \quad state \\ 0 & else \end{Bmatrix}. \tag{6}$$

Failure state in (6) represents any state of the plant outside the predetermined range of controlled state variables.

The original algorithm proved to be sufficient in the case of a hierarchical controller where global optimization was to be achieved through subcontrollers servicing specific sets of state variables. However, when only one subcontroller was applied to a specific set of state variables, a significant steady-state error was observed due to the fact that the binary failure signal implies that any state of the plant inside the predetermined range is "good enough". Therefore, a more informative reinforcement signal *r* is introduced:

$$r[t] = \begin{Bmatrix} -1 & failure \quad state \\ -\dfrac{|x|}{x_{\max}} & else \end{Bmatrix} \tag{7}$$

where x represents the state variable whose value determines whether or not the system reached the failure state. Since in this case the reinforcement signal r is available at each time step, single-step prediction is applied as described in expression (2). In contrast to multi-step prediction in (1), which gives an approximately exponential decay of state scores v if the failure state is not reached, here the state score v permanently acquires values in the interval [-1,0].

Thus, if the learning algorithm according to (5) were implemented, this would gradually lead to a divergence of the ASN parameters because the state score difference $\Delta(\Delta v)$ does not approach zero as in the case of multi-step prediction, which results in a permanent reward of the best action chosen at each time step. Therefore, a modification to (5) is proposed. Instead of applying F^+ at iteration a with the state score of the plant v_1, the expected action F itself is applied. If this action results in an improvement of the state score v_2, the same action is also applied at iteration *b;* otherwise, the ASU stochastically generates an action F', which can be greater than or lesser than the expected action F. After that, F' is applied to the plant, causing the plant to move to a state with a state score v_3. In the case when $F_1 = F_2 = F$ the action difference ΔF is zero, therefore no parameter modification takes place. Otherwise, the learning rule is as follows:

$$\Delta p = \eta \frac{\Delta F \Delta(\Delta v)}{\sigma} \frac{\partial F}{\partial p} \tag{8}$$

where

$$\Delta F \Delta(\Delta v) = (F_1 - F_2)[(v_2 - v_1) - (v_3 - v_2)] = \\ = (F_1 - F_2)[2v_2 - v_1 - v_3)]. \tag{9}$$

In relation (5) parameter modification was achieved through a reward-punishment scheme (Barto, *et.al.*, 1983), whereas in (9) only a punishment scheme was applied. Likewise in contrast to (4), the action difference ΔF is now normalized through deviation σ, which provides a more stable algorithm in the case when the parameters of the ASU, which determine the magnitude of deviation σ, are varied. The flowchart of both learning algorithms is presented in Fig. 4.

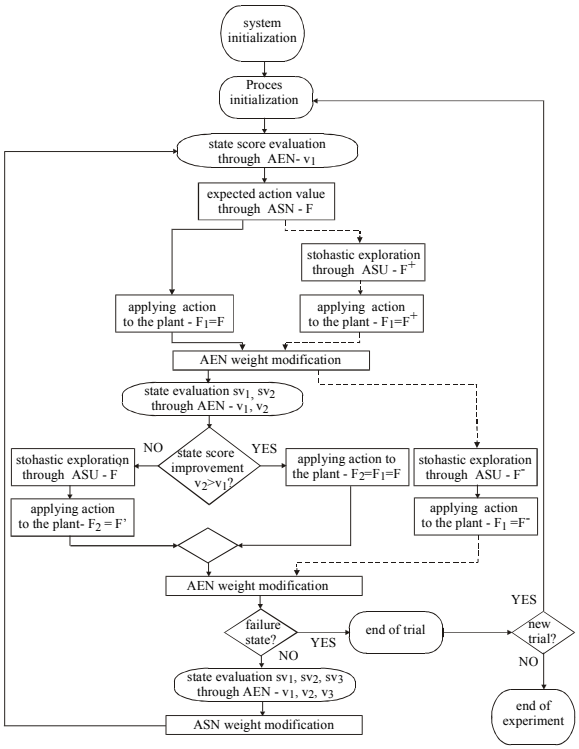

Fig. 4. Flowchart of the modified learning algorithm (solid line) and of the original learning algorithm (dashed line).

The experiment starts with system initialization and consists of trials, each of which finishes when a failure state occurs. Depending on the type of learning algorithm, different actions are chosen (branch conditions).

4. SIMULATION RESULTS

The control architecture based on multi-step prediction and Kandadai and Tien's learning algorithm (CA1) as well as the proposed control architecture with single step prediction (CA2) have been implemented in Matlab/SIMULINK® and tested through simulation experiments. The control problem is balancing of the cart-pole system shown in Fig. 5.

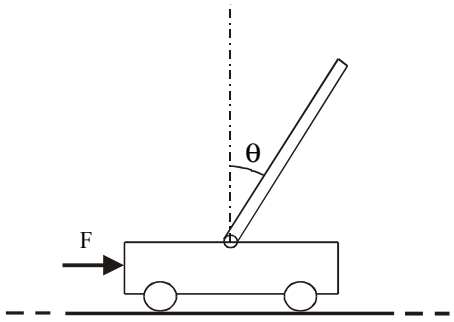

Fig. 5. Cart-pole system.

It is assumed that only the pole position is controlled, whereas the cart is allowed to move on an infinite track. This implies that only state variables θ and $\dot{\theta}$ are taken into consideration.

In case when CA1 is used, the reinforcement signal *r* is of the form:

$$r[t] = \begin{cases} -1 & |\theta| > \theta_{\max} \\ 0 & else \end{cases} \qquad (10)$$

whereas in the case when CA2 is used, it is of the form:

$$r[t] = \begin{cases} -1 & |\theta| > \theta_{\max} \\ -\dfrac{|\theta|}{\theta_{\max}} & else \end{cases} \qquad (11)$$

where $\theta_{\max} = 15^{\circ}$ is chosen. The number of fuzzy rules in both experiments is 7 rules for θ and 5 rules for $\dot{\theta}$.

The simulation results obtained with CA1 and CA2 are shown in Fig. 6 and Fig. 7, respectively. It can be seen that both controllers stabilize the system; however, CA1 with a steady-state error of $\theta \approx 1.5^{\circ}$ and CA2 without a steady-state error. The steady-state error that appears when CA1 is used is due to the binary nature of the reinforcement signal *r*, which indicates that any angle $\theta \le \theta_{\max}$ is satisfactory. Consequently, the state score *v* may become zero before the optimal state is reached ($\theta = 0^{\circ}$), whereas it should have some negative value indicating that the present state of the plant is not optimal (in this case $\theta \approx 1.5^{\circ}$). In addition, the response of the output force F' in the case of CA1 (Fig. 6) shows that the stochastic exploration is persistent, resulting in a much higher control effort rate and faster aging of the actuator of the control system than in the case of CA2.

Fig. 8 shows the results obtained with CA2 in the case when friction in the bearings and measurement noise are taken into consideration. The measurement noise is simulated as white noise. It can be seen that the system is stabilized in a narrow range around the reference state.

5. CONCLUSIONS

The learning algorithm proposed in (Kandadai and Tien, 1997) for generating the knowledge-base of the fuzzy-neural hierarchical controller proved to be insufficient when considering only one subcontroller servicing a specific set of state variables, mainly due to a significant steady-state error. This was the consequence of assuming a binary reinforcement signal leading to the multi-step prediction case.

Therefore, a more informative, continuous reinforcement signal was introduced, leading to the single-step prediction case which also required a modification of the learning algorithm. The modified algorithm showed to be far more accurate in terms of the state-state error. The learning rate was increased together with the robustness of the algorithm to parameter changes. Likewise, the overall energy consumption of the system due to stochastic exploration was decreased because the exploration takes place only when the plant moves to a worse state.

The method introduced was tested in a simulation experiment where the process concerned was the cart-pole system. The controller was able to stabilize the pole in a very narrow range around the reference point. Furthermore, some real-world problems, such as friction in the bearings and noise measurement were successfully compensated.

REFERENCES

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

Berenji, H.R., Khedkar, P. (1992). Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, **3**, No.5, pp.724-740.

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural networks*, **3**, pp.671-692.

Kandadai, R.M., Tien, J.M. (1997). A knowledge-base generating hierarchical fuzzy-neural controller. *IEEE Transactions on Neural Networks*, 8, No.6, pp.1531-1543.

Lin, C.T., Lee, C.S.G. (1996). *Neural fuzzy systems – a neuro-fuzzy synergism to intelligent systems.* Prentice Hall P T R.

Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986). Learning internal representations by error backpropagation.*Parallel distributed processing*, **1**, pp. 318-362, MIT Press, Cambridge MA.

Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, **3**, pp.9-44.

ACKNOWLEDGEMENT

The authors would like to thank to the Ministry of Science and Technology of Republic of Croatia and to the companies Siemens-Croatia and Pliva-Croatia for financial support of this project.

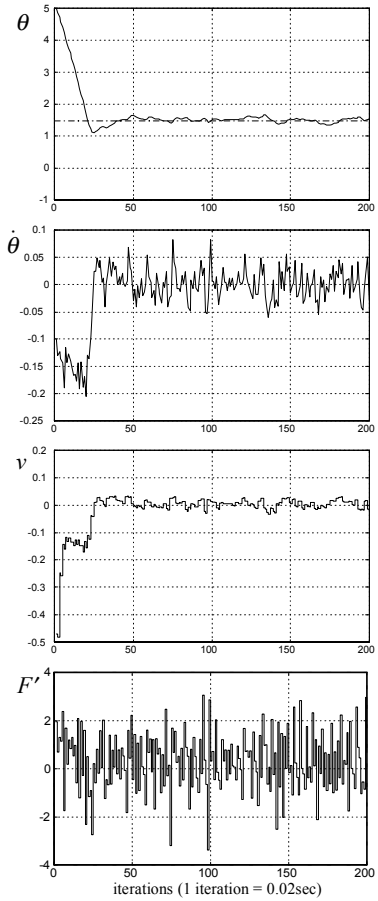

Fig. 6. Simulation results on the cart-pole balancing system obtained with CA1.

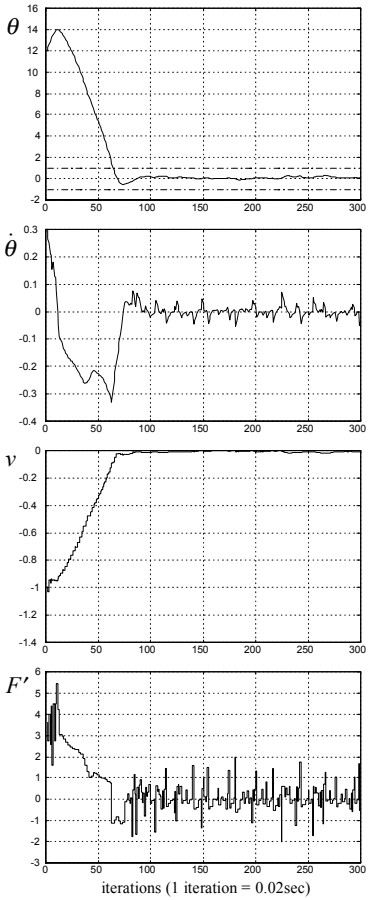

Fig.7. Simulation results on the cart-pole balancing system obtained with CA2.

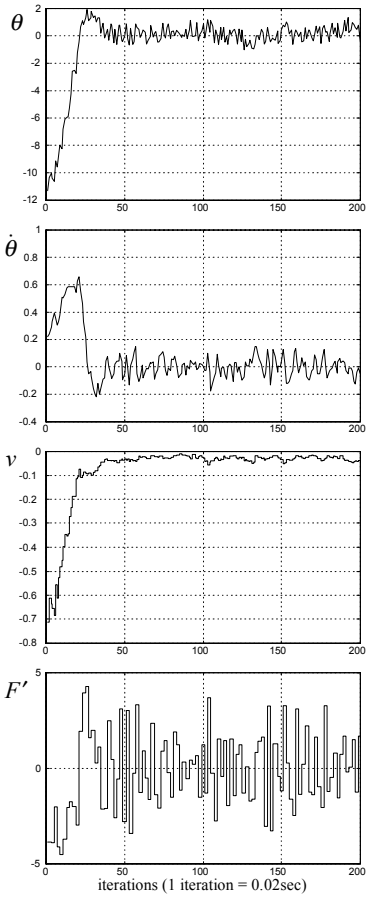

Fig. 8. Simulation results on the cart-pole balancing system obtained with CA2 in the presence of friction in the bearings and measurement noise.