NASA-TM-111486

Using Fuzzy Logic for Performance Evaluation in Reinforcement Learning

Hamid R. Berenji
Intelligent Inference Systems Corp.
AI Research Branch, MS: 269-2
NASA Ames Research Center
Mountain View, CA 94035

Pratap S. Khedkar[1]
CS Division, Department of EECS,
University of California at Berkeley,
Berkeley, CA 94720
khedkar@cs.berkeley.edu

Abstract

Current reinforcement learning algorithms require long training periods which generally limit their applicability to small size problems. A new architecture is described which uses fuzzy rules to initialize its two neural networks: a neural network for performance evaluation and another for action selection. This architecture is applied to control of dynamic systems and it is demonstrated that it is possible to start with an approximate prior knowledge and learn to refine it through experiments using reinforcement learning.

1 INTRODUCTION

Reinforcement Learning (RL) can be used in domains where learning has to be done without the presence of a direct supervisor and through a distal teacher. Unlike supervised learning, an explicit error signal is not assumed in RL and external reinforcement may be delayed. In GARIC [1], RL is

[1]Supported by NASA grant NCC-2-275 and MICRO

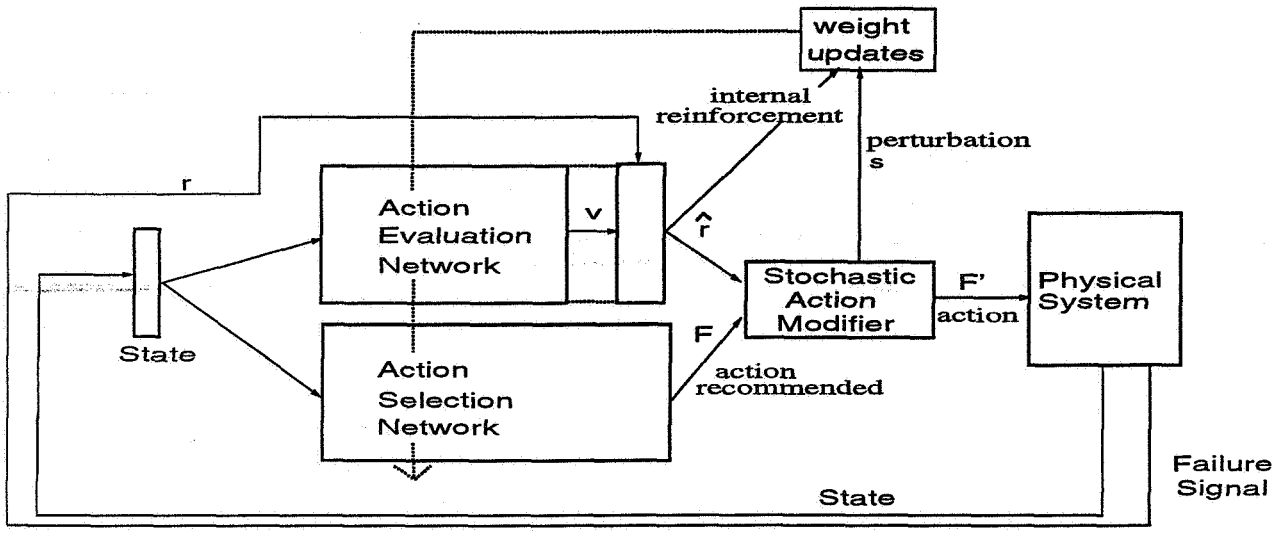

Figure 1: The architecture of GARIC

combined with Fuzzy Logic Control (FLC) [2] to refine the knowledge base of a controller. GARIC is composed of three main elements: an Action Selection Network (ASN) which maps the state to an action using fuzzy control rules; an Action-state Evaluation Network (AEN) which evaluates the action and the resulting system state; and a Stochastic Action Modifier (SAM) which explores the search space for possible actions (see Figure 1). In GARIC, fuzzy inference is used only in the ASN to incorporate prior knowledge as well as to handle continuous input-output without artificial discretizing. The AEN remained a two-layer feed forward neural net, which starts with random weights, an ad hoc architecture, and which may not be able to handle complex tasks.

In this paper, concentration is on using fuzzy inference in the design and operation of the evaluation network. Specifically, the problem of how to use prior knowledge to design the architecture is addressed getting a head start on the way of learning to evaluate. Fuzzy rules are used to represent the heuristic knowledge of state evaluations.

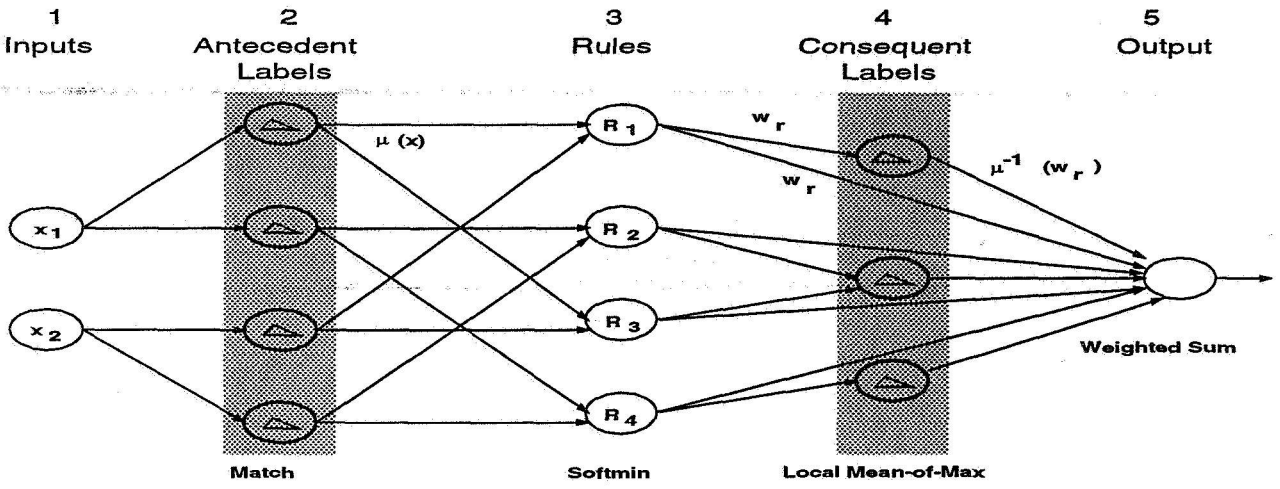

Figure 2: The Action Evaluation Network

2 NETWORK ARCHITECTURE

Earlier, Anderson [3] used conventional neural nets to implement both the ASN and AEN, but since these were initialized randomly, learning needed a large number of trials. In GARIC [1], the ASN was initialized using approximate rules, which were used to drive a neural net implementing fuzzy inference. The incorporation of heuristic knowledge led to substantial reduction in learning time. Here, this principle is further extended by being applied to the AEN (the evaluation critic) and by using fuzzy rules that will help in computing the goodness of a state.

To build in fuzzy rules into the net, some modifications in its structure are required. Both the ASN and AEN will now have similar architectures, and each is based on some initial rule base. The structure of the net consists of 5 layers, connected in feedforward fashion, and shown in Figure 2.

Layer 1 is the input layer and performs no computation.

A Layer 2 node represents one possible linguistic value of one of the input variables. It computes $\mu_L(x)$, and outputs using the clause: if x is L in their if part.

Layer 3 implements the conjunction of all the antecedent conditions in a rule using the softmin operation. There is one node per rule here; its inputs

come from all its antecedents, and it produces w_r, the degree of applicability of rule r.

A Layer 4 node represents a consequent label. Its inputs come from all rules which use this consequent label. For each w_r supplied to it, this node computes the corresponding output action as given by rule r.

A Layer 5 node combines the recommended actions from all the rules, using a weighted sum, the weights being the rule strengths w_r. In the AEN, a state score v is produced (see [1] for more details). Learning modifies weights into Layers 2 and 4 only, the others being fixed at unity.

3 LEARNING IN THE AEN

The learning algorithm is largely determined by the choice of the objective function used by each component for optimization. Two such choices and the corresponding results are described. For both policies, both AEN and ASN learn simultaneously as per the learning cycle outlined in Figure 3. Also for both policies discussed here, AEN outputs v which is then combined with an external reinforcement r to produce $\hat{r}$.

In policy 1, the ASN retains its earlier objective of maximizing the state-score v. However, the AEN tries to maximize the internal reinforcement $\hat{r}$, since $\hat{r} \approx 0$ is a good prediction of failure and a high $\hat{r}$ otherwise is equivalent to moving to better states. Tuning the AEN parameters to attain this is done by computing $\partial\hat{r}/\partial v$ from

$$\hat{r}[t+1] = \begin{cases} 0 & \text{starting state ;} \\ r[t+1] - v[t,t] & \text{failure state;} \\ r[t+1] + \gamma v[t,t+1] - v[t,t] & \text{otherwise.} \end{cases} \tag{1}$$

Then a gradient descent method leads to,

$$\Delta p = \eta \, \frac{\partial \hat{r}}{\partial v} \cdot \frac{\partial v}{\partial p}, \tag{2}$$

where $\frac{\partial \hat{r}}{\partial v} \approx d\hat{r}/dv = (1-\gamma)+\gamma(d^2v)$, assuming the derivative doesn't depend on r. The second derivative of v is approximated by the finite difference $v[t]-2v[t-1]+v[t-2]$, and only the sign $\frac{\partial \hat{r}}{\partial v}$ is used so that noise is reduced. The term $\partial v/\partial p$ is the dependence of the net output on its parameters (the

load-state();
v_{t-1} = evaluate-state(); /* *AEN:1* */
apply-action(action = SAM(select-action(),$\hat{r}_{t-1}$)); /* *ASN:1* */
load-state();
v_t = evaluate-state(); /* *AEN:2* */
compute $\hat{r}_t$, gradients;
modify-parameters(); /* *learn as per data in AEN:1 and ASN:1* */

Figure 3: Steps in a learning cycle

centers and spreads of the membership functions) and can be easily computed using a backpropagation-like scheme [1].

In policy 2, a different objective function can be used. If the future, discounted reward be equal to $\sum_{j>0} \gamma^{j-1} r_{t+j}$, then v may be interpreted as a truncation of this series to 1 or 2 terms. For good prediction, $v(t)$ should closely approximate $r(t+1)$. Thus minimizing the error $(v_t - r_{t+1})^2$ is needed. Learning in both AEN and ASN is geared towards this same objective.

4 RESULTS

4.1 CART-POLE BALANCING

In this problem a pole is hinged to a cart which moves along one dimension. The control tasks are to keep the pole vertically balanced and the cart within the track boundaries. The displacement and velocity of the cart $(x, \dot{x})$, and of the pole $(\theta, \dot{\theta})$ is the system state. The action is the force F to be applied to the cart. A failure occurs when $|\theta| > 12°$ or $|x| > 2.4$ m, whereas a success is when the pole stays balanced for 100000 timesteps ($\approx$ 33 minutes of real time). $\hat{r}$ is calculated using $\gamma = 0.9$. Also, half-pole length = 0.5 m, pole mass = 0.1 kg, cart mass = 1.0 Kg. A trial lasts from an initial state to success or failure.

The design of the initial ASN rule base is from [4, 5], and results in 9 and 4 rules for controlling the pole and cart respectively. So the architecture has 4 inputs, 14 units in layer 2 (the number of antecedent labels), 13 units in layer 3 (the number of rules), 9 units in layer 4 (the number of consequent

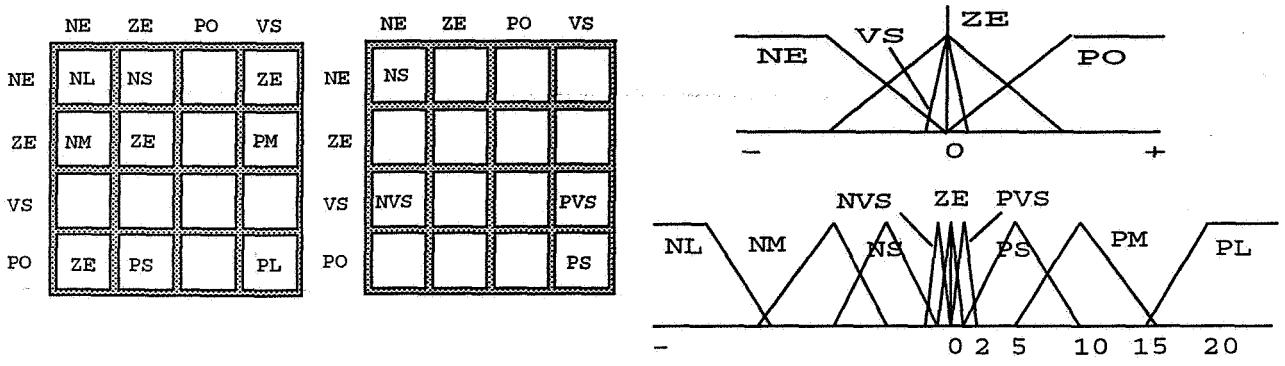

Figure 4: The 9+4 rules for the ASN; four qualitative labels for each input, and nine labels for Force.

labels) and one output (force) as shown in Figure 4. The AEN is started with 10 rules, with 4,12,10,3, and 1 nodes in its 5 layers respectively. All the rules and membership functions involved are shown in Figure 5. The resulting input-output functions are shown in Figure 6.

The experiments performed are of three types: (a) changes of tolerance and physical system values, (b) damage to parameters of the membership functions, (c) changes to the rule base reflecting different granularity. The damages to parameters can be for the AEN or ASN or both. Learning is by Policy 1 or 2. In the following figures, each graph shows the first two trials (up to 6 sec), and the first and last 6 sec of the final (successful) trial. Both policies are considered. Some runs are shown and explained in Figures 7,8,9 for Policy 1 and Figures 10,11, 12 for Policy 2. The learning is quicker by about one or two orders of magnitude, when compared to a randomly started AEN. Overall, Policy 1 is better, learning faster and shifting labels consistently.

4.2 BACKING UP A TRUCK

This problem involves backing up a truck so that it reaches a loading dock at a right angle. The two inputs are the x-coordinate of the rear of the truck, and its angle (ϕ) to the horizontal. The output is the steering-angle (θ). The ASN rules are from [6], whereas the AEN rules were approximately designed

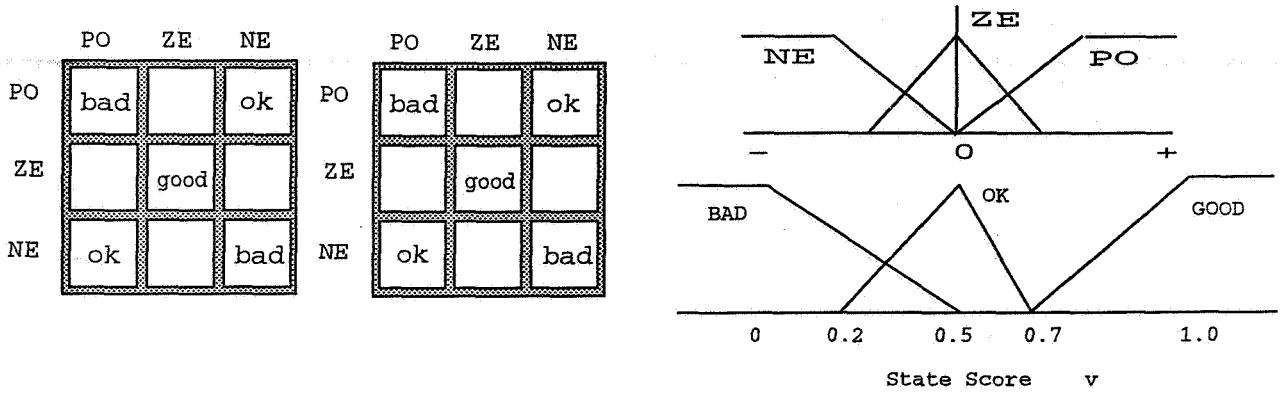

Figure 5: The 5+5 rules for the AEN, followed by the membership functions (3 each for the 4 input and 1 output variable).

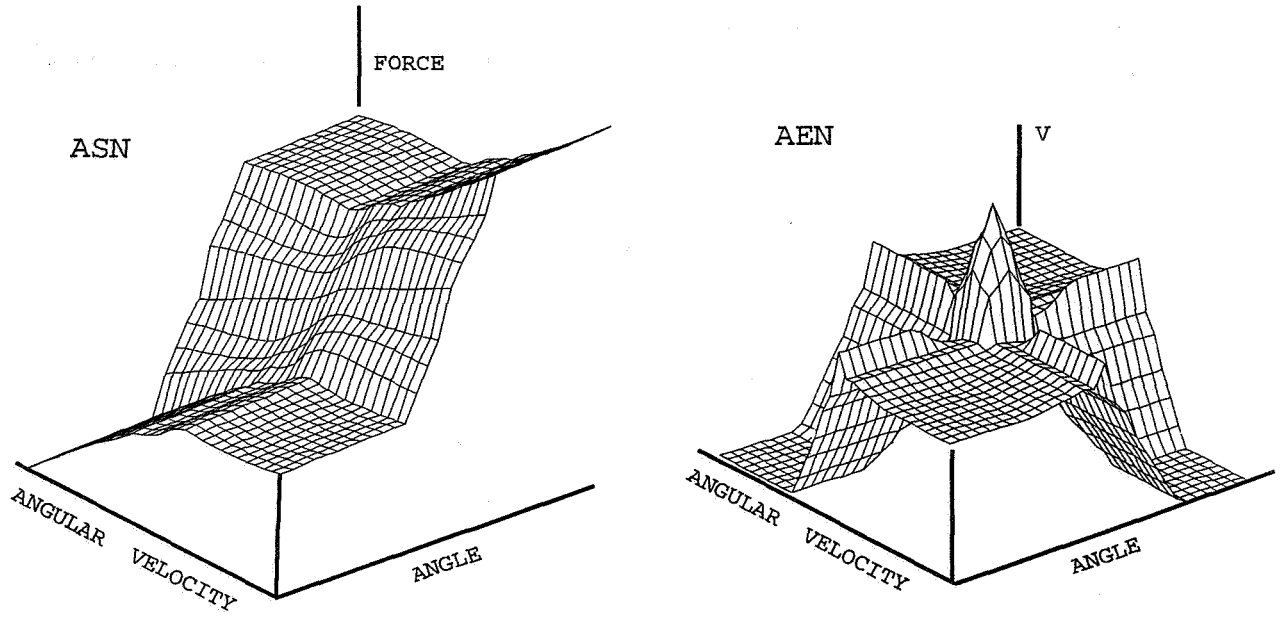

Figure 6: I/O surfaces implemented.

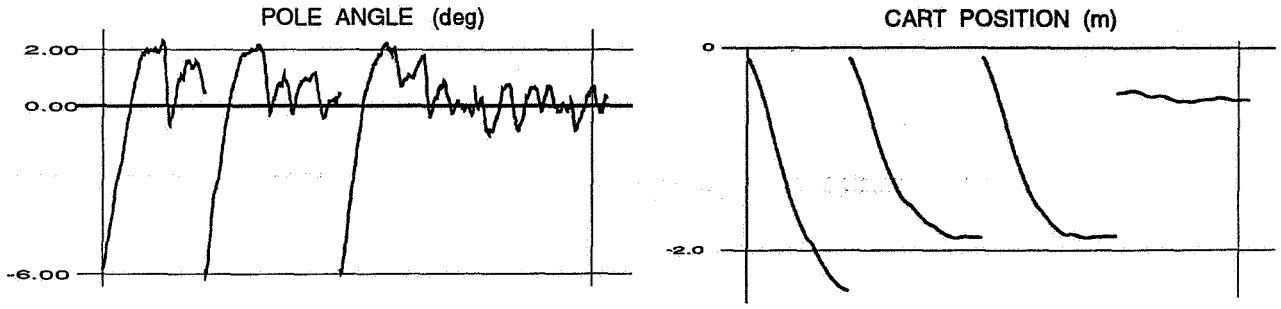

Figure 7: Policy 1, 3 antecedent AEN labels, 2 consequent AEN labels and 3 consequent ASN labels damaged. Start position = –0.1. Learning took 3 trials.

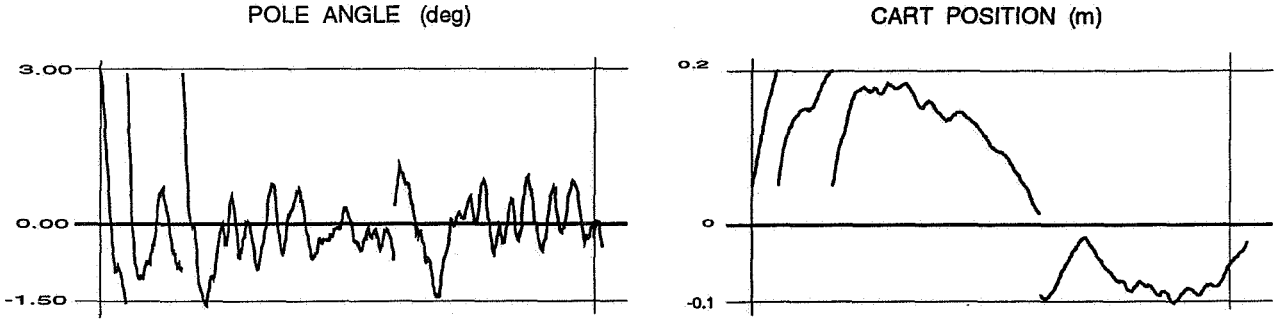

Figure 8: Policy 1, Tolerance changes: $|\theta|$: 0.2 $\rightarrow$ 0.1, $|x|$: 2.4 $\rightarrow$ 0.4, l : 0.5 $\rightarrow$ 0.4, Start position = 0.05. Learning took 3 trials.

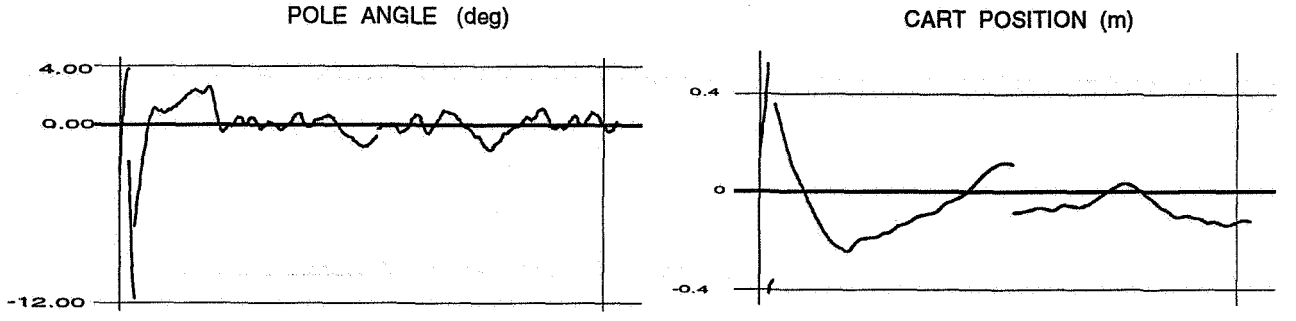

Figure 9: Policy 1, $|x| : 2.4 \rightarrow 0.5$, AEN: 3 antecedent, 1 consequent labels changed, random start-positions. Learns in 4 trials.

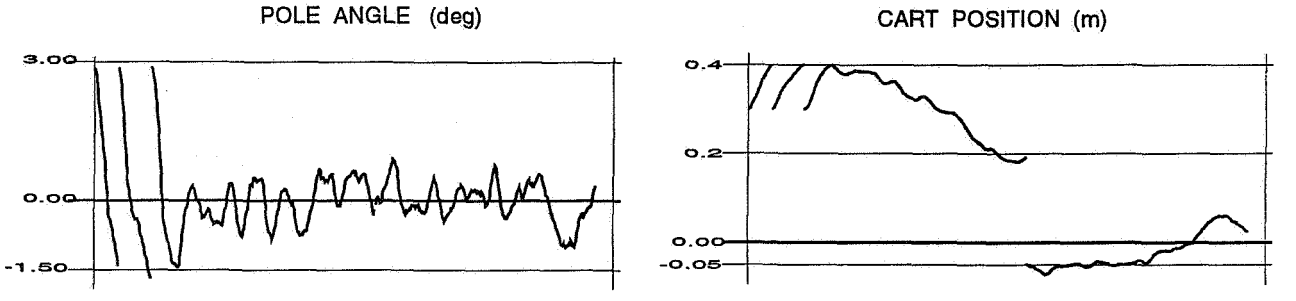

Figure 10: Policy 2, Same change as Figure 8, learnt in 18 trials.

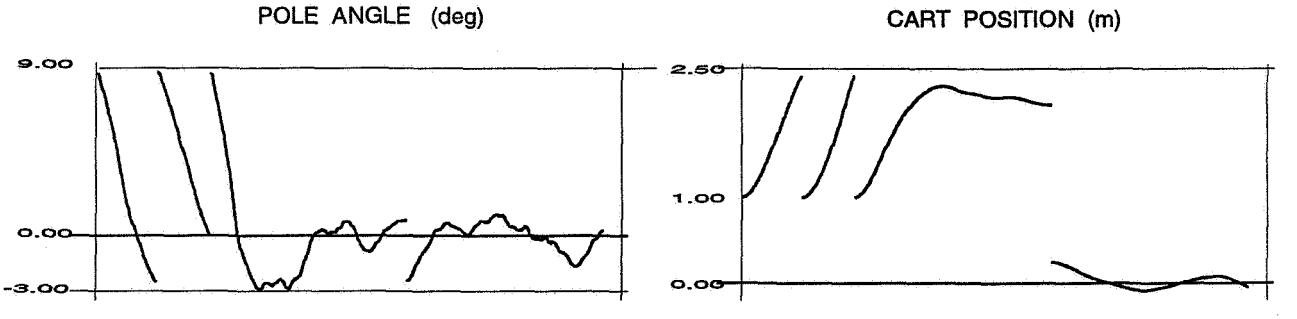

Figure 11: Policy 2, good and bad both changed to center at -1. good was shifted to 0.

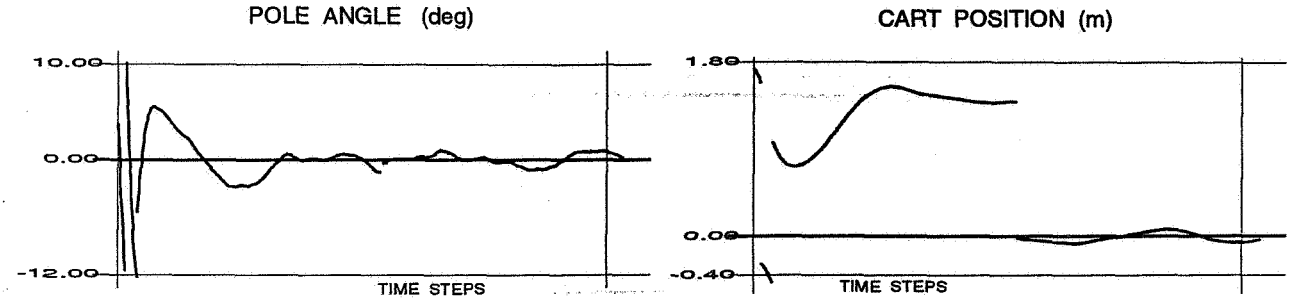

Figure 12: Policy 2, $m_{cart} = 2$ kg (from 1 kg). Random starts, learnt in 4 trials.

based on simple considerations and are given in Figure 13. The evaluation here is based on the same inputs x and ϕ, and the basic surface generated by the AEN is quite similar to the one used in the pole-balancing problem. Since it is desirable for the truck to be centered and pointing straight down, $(50, 90)$ is a *good* state. When x is left of center, an angle less than 90 is desirable, since it can then approach the center line quickly. However, and angle greater than 90 is a bad state, since more maneuvering is required. Using these considerations, five simple rules were devised for the AEN.

The GARIC architecture for this problem has 2,12,35,7 and 1 units in the ASN layers, and 2,6,5,3 and 1 units in the AEN layers respectively. The initial ASN rulebase assumes sufficient y-coordinate clearance.

The results presented in Figures 14 and 15 are from the older scheme when the AEN was a randomly initialized neural net. In Figures 16 and 17, we see results when the AEN is initialized using the rules discussed before. The ASN uses the same 35 rules in all cases. The curves show the pre- and post-learning paths of the rear-end of the truck.

An interesting phenomenon was observed when the damage was too great to rectify. Since $\hat{r}$ maximization is the goal, the system usually manages to achieve it via correction of the ASN labels. However, when the damage here is such that correction is not done quickly, the gradient descent mechanism begins to act with increasing pressure on the AEN output labels, specifically, the label "good". It is the definition of these labels that plays a key role in defining the value of v, and therefore $\hat{r}$. In fact, the system discovers that steadily increasing the value of v by pushing the label "good" to the right, is

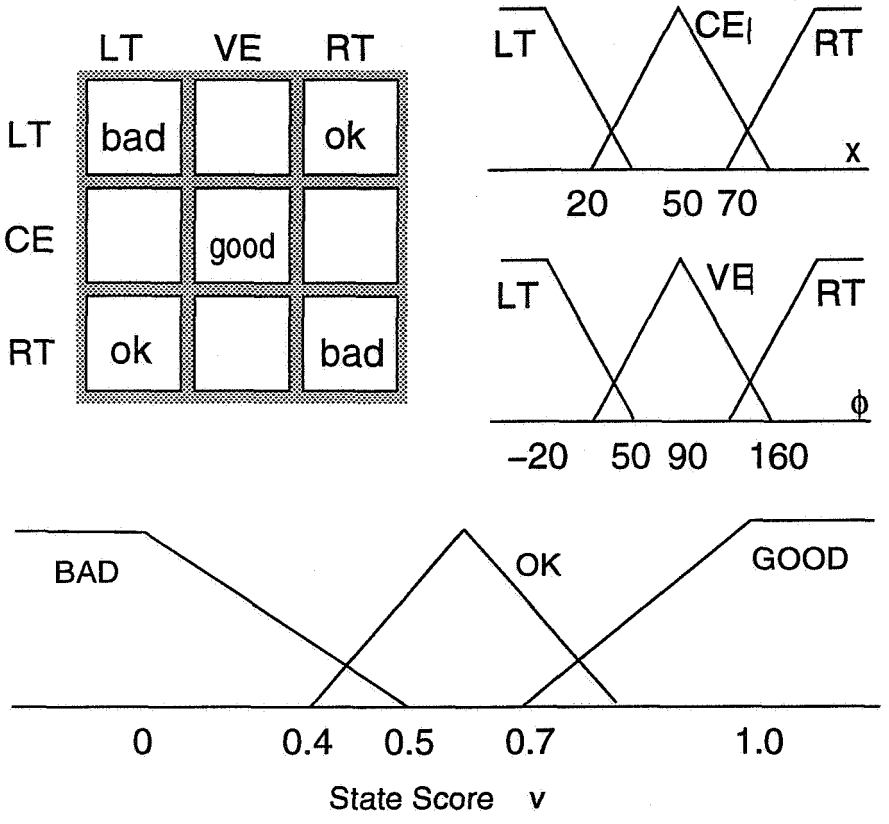

Figure 13: The 5 rules which evaluate the state for the truck-docking problem, and the 9 membership functions needed (3 per variable).

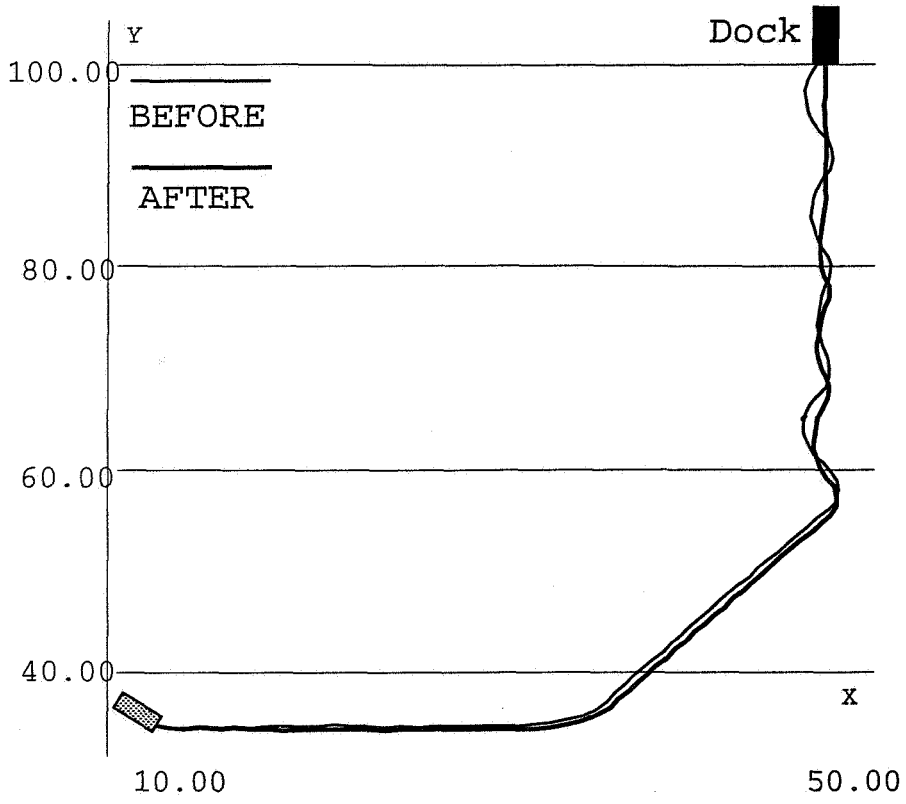

Figure 14: Learning to back up a truck when AEN is not initialized with rules and linguistic labels are incorrect.

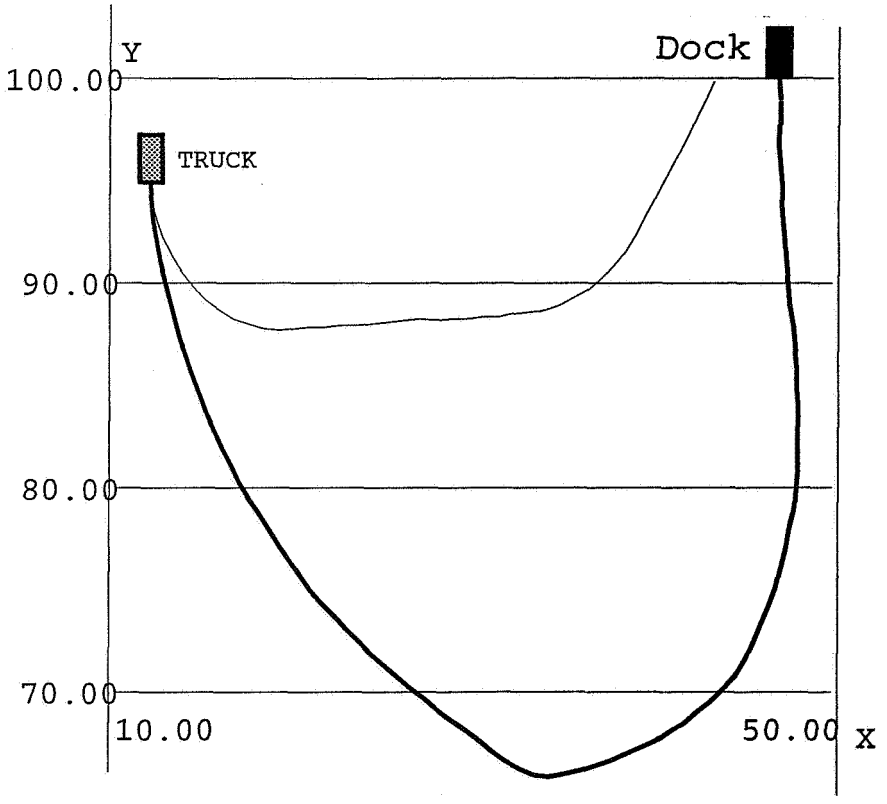

Figure 15: Learning to back up a truck when AEN is not initialized with rules and inference is done with incomplete knowledge (y-coordinate not known).

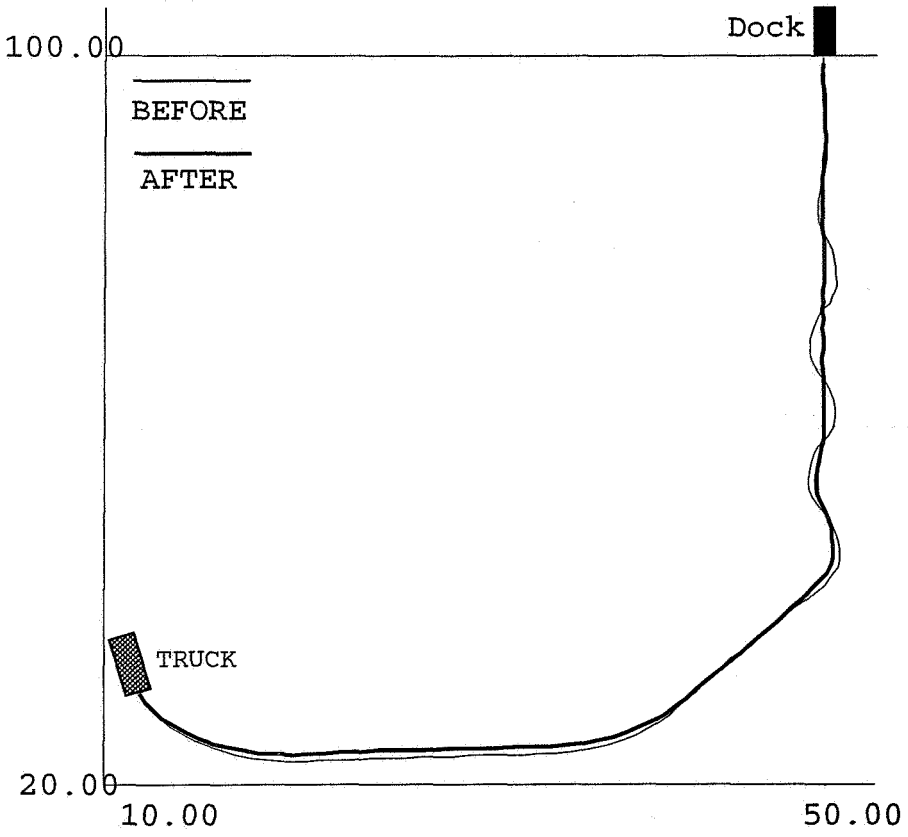

Figure 16: Learning to back up a truck when the AEN is initialized using fuzzy rules, then extensive label damage is quickly repaired.

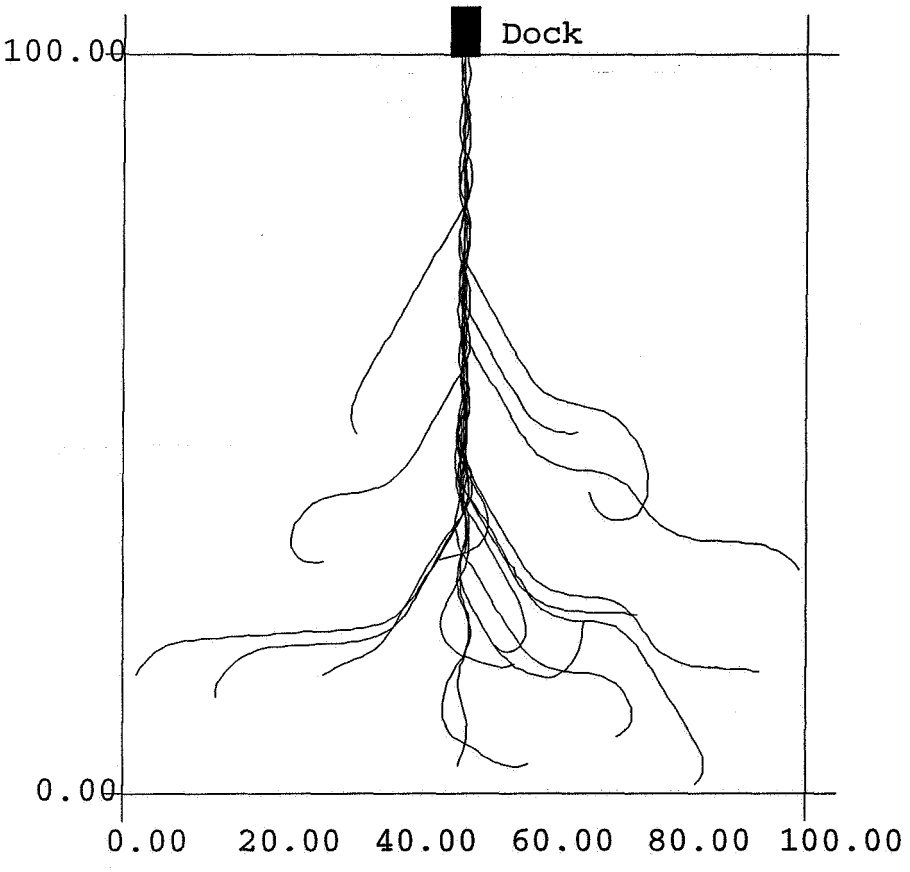

Figure 17: Learning to back up a truck when the AEN is initialized using fuzzy rules, then learning occurs even when the start-position after each failure is randomly chosen.

a better way to achieve high $\hat{r}$, at least in all those time steps which are not labeled as failure. Therefore, except in the instant where the truck actually falls off the platform, the system redefines "good" so as to appear to be doing well even when it is not learning in the desired way. This phenomenon can be eleminated by either hard-limiting the positions of the AEN labels, or by reducing the learning rate on them (as compared to the η for the ASN). This may also be the result of choosing $\hat{r}$ as the objective function rather than some other measure. Of course, choosing v in its place (as was done for the ASN in GARIC earlier) would lead to a similar problem. Since absolute scales for both v and $\hat{r}$ are quite meaningless, restricting them to any arbitrary range is permissible, so a hard-limit may be a reasonable solution here.

5 CONCLUSION

A nonrandom initialization of the neural networks, if guided by heuristic knowledge will substantially speed up learning. Extensive retraining is unnecessary if there are tolerance/parameter changes. A unified approach is shown by which a few simple, heuristic and imprecise rules can be directly built into a neural network as a starting configuration, and all subsequent tuning is performance-driven and automated. By doing this, we gain substantially in learning speed and achieve a uniform integration of RL and fuzzy inference. By changing the rules, the state of the system is kept within a particular region of the state-space. More informative reinforcement signals can be easily incorporated. For complex tasks, inclusion of prior knowledge can have a significant effect on learning speed. This hybrid method offers a broader scope by combining the robustness of fuzzy logic and the learnability of neural nets.

References

[1] H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5), 1992.

[2] H. R. Berenji. Fuzzy logic controllers. In R. R. Yager and L.A. Zadeh, editors, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, pages 69–96. Kluwer Academic Publishers, 1991.

[3] C. W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[4] H. R. Berenji, Y. Y. Chen, C. C. Lee, S. Murugesan, and J. S. Jang. An experiment-based comparative study of fuzzy logic control. In *American Control Conference*, Pittsburgh, 1989.

[5] H.R. Berenji, Y.Y. Chen, C.C. Lee, J.S. Jang, and S. Murugesan. A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems. In P.P. Bonissone, M. Henrion, L.N. Kanal, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence: Volume VI, in the series Machine Intelligence and Pattern Recognition*, pages 331–343. Elsevier, North-Holland, 1991.

[6] B. Kosko. *Neural Networks and Fuzzy Systems.* Prentice Hall, 1992.