Learning fuzzy control laws for the inverted pendulum

R.J. Stonier
Mathematics & Decision Science
Central Qld University
Rockhampton, Australia 4702

A.J. Stacey
Dept. of Mathematics
R.M.I.T
Melbourne, Australia 3000

C. Messom
Engineering Dept.
Singapore Polytechnic
Singapore 139651

Abstract

In this paper we will examine the problem of learning an efficient fuzzy logic rule set for the control of the inverted pendulum (with nonlinear dynamics) using an evolutionary algorithm. In particular we compare a two layered rule set with a single fuzzy logic rule set. Furthermore we look at the effect that different choices of objective function (in the evolutionary algorithm) have on the rule sets that are learnt.

1 Introduction

The problem of controlling an inverted pendulum or simulated pole-cart system has a long history with approaches that use linear and nonlinear dynamics and include both classical and fuzzy logic control techniques, see for example [1, 2, 3, 4, 5, 6] and references therein. It has already been demonstrated that this system can be controlled by a fuzzy logic controller the rules for which can be learned by a genetic algorithms. However it is possible to start with fuzzy logic rule sets that have different structures, for instance a single rule set or a multilayered rule set. It is also known that when the objective function is changed in an optimization problem it can lead to different solutions. These variations can all lead to different rule sets which may be more or less effective in controlling the system. It is an initial investigation of these possibilities that the paper examines. In particular we compare a multilayered, two-level rule set with a single fuzzy logic rule set and look at the effect of including different objectives in the fitness function. Although results will be given for the inverted pendulum there are obvious implications for other systems.

2 Inverted Pendulum Sytem

The nonlinear system to be controlled consists of the cart and a rigid pole hinged to the top of the cart. The cart is free to move left or right on a straight bounded track and the pole can swing in the vertical plane determined by the track. It is modelled by:

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= U + m\ell(\sin(x_3)x_4^2 - \dot{x}_4\cos(x_3))/(M+m) \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= \frac{g\sin(x_3) + \cos(x_3)(U - m\ell x_4^2\sin(x_3))/(M+m)}{\ell(4/3 - m\cos(X3)^2/(M+m))}
\end{aligned}$$

where x_1 is the position of the cart, x_2 is the velocity of the cart, x_3 is the angle of the pole, x_4 is the angular velocity of the pole, U is the control force on the cart, m is the mass of the pole, M is the mass of the cart, ℓ is the length of the pole, and g is gravitational acceleration. The control force is applied to the cart to prevent the pole from falling while keeping the cart within the specified bounds on the track. We take $m = 0.1kg$, $M = 1kg$, $\ell = 0.5m$, $g = 9.81ms^{-2}$, with state limits: $-1.0 \leq x_1 \leq 1.0$, & $-\pi/6 \leq x_3 \leq \pi/6$.

Our goal is to determine fuzzy controllers necessary to stabilise the system about the unstable reference position $\mathbf{x} = \mathbf{0}$ as quickly as possible, whilst maintaining the system within the specified bounds given above.

3 Fuzzy Rule sets

Two structures for the fuzzy rule set are given, the first a single rule base case. All four of the variables x_1 to x_4 are used as inputs to the fuzzy rule set to produce a control output U. For convenience all input and output domains were normalised to lie in $[-1, 1]$. Actual values are re-established for the integration of the state equations. Each domain region for x_i was divided into 5 overlapping intervals and assigned linguistic values NB - Negative big, NS - negative small, CE - Center, PS - Positive small, PB - Positive big, which we encode numerically as integers from 1 to 5 respectively. The output variable U is divided into 7 overlapping regions, five as above and appropriately

two others, NM - Negative medium, and PM - Positive medium, with integer encoding 1 to 7. All fuzzy membership functions are assumed to be triangular.

The fuzzy rule base of 625 rules may be represented by a $5 \times 5 \times 5 \times 5$ array $M(i, j, k, \ell)$, with indices referring to the variables x_1 to x_4, and having the values 1 to 5. $M_(i, j, k, \ell)$ will then take values from 1 to 7. As illustration, the rule defined by $M(1, 2, 5, 4) = 2$ reads: If $x_1 = \mathbf{NB}$ and $x_2 = \mathbf{NS}$ and $x_3 = \mathbf{PB}$ and $x_4 = \mathbf{PS}$ Then $U = \mathbf{NM}$.

In the two fuzzy knowledge base case the first knowledge base has the two inputs, x_1 (position of the cart) and x_2 (velocity of the cart) to produce as output an offset angle W. The offset angle is then added to x_3 (the current angle of the pole). This updated value of x_3 and x_4 (the angular velocity of the pole) are used as input in the second knowledge base to produce the control output U.

The input and output fuzzy sets have the same conventions as defined above but now the two fuzzy rule bases are stored in the two arrays $M_1(i, j)$, with indices referring to x_1, and x_2 producing a linguistic value for the offset angle W as its output, and $M_2(1, 2)$ with indices referring to x_3, and x_4 and producing a linguistic value for the control output U. There are a total of 50 fuzzy rules in this structure.

4 Evolutionary learning

We use Evolutionary algorithms [7] whose strings are integer encoded to learn the fuzzy control rules in these fuzzy knowledge bases. This heuristic search technique maintains a population of strings $P(t) = \{\mathbf{x}_1(t), \ldots \mathbf{x}_n(t)\}$ at iteration t to the next $t + 1$.

Each string encodes a rule base(s) which is a possible solution to the control problem. In creating the new population, proportional selection, a variant of arithmetic crossover, and a modified mutation operator are used. The strings are formed by storing the elements of the array(s) above in a one dimensional array. For example the array M can be stored as a single linear array of 625 elements by linearising on the fastest changing index ℓ. The arrays M_1 and M_2 can each be stored as a single 25 element linear array and then concatenated to form a linear array of 50 elements. A second integer is appended to each rule in the string producing a two dimensional array, ie 50×2 for our example. The second integer is used to store the clipping level applied to the output fuzzy set and is used in the crossover operator.

The modified arithmetic crossover algorithm is encapsulated in the code below. It says that if a given rule (j) in parent1 is not used and the corresponding rule in parent2 is used then both children inherit the rule that is used. Likewise, the reverse holds. If both rules are used or not used, we perform the traditional type of arithmetic crossover defined by a constant α with $0 < \alpha < 1$, except that rounding is used to ensure that integer entries are maintained. The purpose of the crossover algorithm is to ensure that information within the rule structure is passed to the children. In essence it is performing in some sense, fuzzy amalgamation [8], on the fly at a local level.

```
for j := 1 to length_of_string do
begin
  if((parent1[j,2]=0) and (parent2[j,2]<>0)) then
  begin
    child1[j,1] := parent2[j,1];
    child2[j,1] := parent2[j,1];
    child1[j,2] := parent2[j,2];
    child2[j,2] := parent2[j,2];
  end
  else if ((parent1[j,2]<>0)
                      and (parent2[j,2]=0)) then
  begin
    child1[j,1] := parent1[j,1];
    child2[j,1] := parent1[j,1];
    child1[j,2] := parent1[j,2];
    child2[j,2] := parent1[j,2];
  end
  else
  begin
    child1[j,1] := round(alpha*parent1[j,1]
                      + (1-alpha)*parent2[j,1]);
    child2[j,1] := round((1-alpha)*parent1[j,1]
                      + alpha*parent2[j,1]);
    child1[j,2] := round(alpha*parent1[j,2]
                      + (1-alpha)*parent2[j,2]);
    child2[j,2] := round((1-alpha)*parent1[j,2]
                      + alpha*parent2[j,2]);
  end;
end;
```

The mutation operator takes a probability of mutation and is defined as: If $\mathbf{x}_i[k, 1]$ lies between (but not including) 1 and 7 it is mutated up or down by one with equal probability; if 1 it is mutated to 2, and if 7 it is mutated to 6. Further an 'elite' option is used in developing the new population from the old and prescaling used to enhance convergence.

The fitness of a given string in the population can be evaluated as follows. Given an initial condition of the system we can decode each string $\mathbf{x}_i$ into a fuzzy logic controller and apply this to the system by integrating the state equations by an Euler or Runge-Kutta algorithm with step size 0.02 over a sufficiently long

time interval $[0, T]$. An objective function f_i to be minimised, is then calculated based on some measures of the behavior of the system over the time interval. Such measures might include, the accumulated sum of normalised absolute deviations of x_1 and x_3 from zero, the average deviation from vertical, the average deviation from the origin or $T - T_S$ where T_S, the survival time, is taken to mean the total time before the pole and cart break some bounds. A penalty of 1000 is added to the objective if the final state breaks the following bounds $|x_1| \leq 0.1$, $|x_2| \leq 0.1$, $|x_3| \leq \pi/24$, $|x_4| \leq 3.0$. The problem is then converted to one of maximisation by defining the fitness of each string to be: $F_i = Maxfit - f_i$ if $Maxfit - f_i > 0$ or $F_i = 0$ otherwise, where $Maxfit$ is the maximum value of the objective function in the current population. This procedure will find a rule set that applies to a single initial configuration. The current objective function has the form:

$$f_i = \omega_1 f_1 + \omega_2 f_2 + \omega_3 f_3 + \omega_4 f_4 + \omega_5 f_5 \quad \text{with}$$

$$f_1 = \frac{1}{N}\sum_1^N \frac{|x_1|}{x_{max}}, \quad f_2 = \frac{1}{N}\sum_1^N \frac{|x_2|}{\dot{x}_{max}}, \quad f_3 = \frac{1}{N}\sum_1^N \frac{|x_3|}{\theta_{max}},$$

$$f_4 = \frac{1}{N}\sum_1^N \frac{|x_4|}{\dot{\theta}_{max}}, \quad \text{and} \quad f_5 = \frac{1}{N}(TS - N),$$

where $x_{max} = 1.0$, $\theta_{Max} = pi/6$, $\dot{x}_{max} = 1.0$, $\dot{\theta}_{Max} = 3.0$, N is the number of iteration steps, $T = 0.02 * TS$ and ω_k are selected positive weights. The first and second terms determine the accumulated sum of normalised absolute deviations of x_1 and x_3 from zero and the third term when minimised, maximises the survival time.

5 Results

For the two knowledge base case with $\omega_1 = 3000.0$, $\omega_2 = 0.0$, $\omega_3 = 3000.0$, $\omega_4 = 0.0$, $\omega_3 = 5000.0$, population size 100, $\alpha = 0.7$ and schedule for mutation probability reducing from $p_m = 0.7$ to 0.001 over 1000 generations, we obtained similar quick convergence to two "best" knowledge bases. The following diagram shows the x error for the best solution obtained from the initial configuration $(0.5, 0.0.0.01, 0.0)$. (Similar for θ, $\dot{x}$ and $\dot{\theta}$.) We observe that within the first 200 iterations, (4.0 time units) the system has converged to within the specified bounds. It was found that the decreasing schedule of mutation probability ensured faster convergence, further it increased the probability of successful application of the algorithm

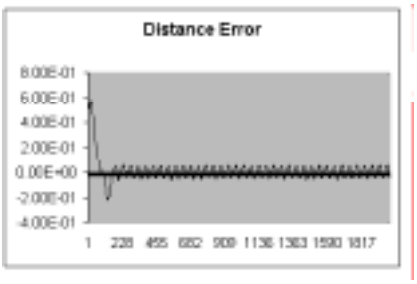

Figure 1: Errors in x.

with differing initial random populations and a range of different initial configurations (see below). Differing parameters ω_k, yielded substantially different convergence. For the example above, the objective function reduced to 280.2 quickly around 400 generations but would not reduce further in 1000 generations. A change to $\omega_1 = 300.0$, $\omega_2 = 50.0$, $\omega_3 = 50.0$, $\omega_4 = 0.0$, and $\omega_5 = 300.0$, resulted in convergence to 37.8 in less than 500 generations and continued decrease to 29.3 in 1000 generations.

For the single knowledge base case with, $\omega_1 = 1000.0$, $\omega_2 = 0.0$, $\omega_3 = 1000.0$, $\omega_4 = 0.0$, $\omega_3 = 5000.0$, population size 60, $\alpha = 0.7$ and schedule for mutation probability reducing from $p_m = 0.6$ to 0.001 over 400 generations, we obtained convergence to a good set of rules (416 used) for the fixed initial state above within around 600 generations. The objective function reduced to 29.38 in 1000 generations. Further after 400 generations $|x_1| < 0.01$. Similar good bounds were obtained on the other variables Showing that the single knowledge based yielded better rules as was expected.

To find a fuzzy controller that will control the system over a wide range of initial configurations we select a grid of N initial configurations in the workspace, then either:

(i) We can form a set of N fuzzy logic knowledge bases, one for each initial configuration as shown above, and amalgamate these by averaging the outputs for each rule in the knowledge bases [9, 10] to produce a single fuzzy logic knowledge base that can be used effectively for a wide range of initial configurations not just those defined by the grid; or

(ii) We can determine a single fuzzy knowledge base by a single evolutionary learning algorithm by making appropriate modifications to the algorithm. See [8] for application in target tracking.

Due to the limited space requirement, we report here briefly on our initial investigations using (ii) .

First with a significantly increased population, typically around 400 for $N = 256$, fitness evaluation for each string is taken as an average of the fitness evaluated on a random selection of distinct initial configurations from the grid, typically we used 10. This was done in order to ensure objective performance across the grid. Second, in the elitist strategy, we typically made 4 or 5 copies each of the top 5 "best" strings in passing from the old generation to the new. These strings when passed across have their fitness values again re-evaluated as above. This many were carried across to ensure that information in the previous population was not lost completely in re-evaluation of fitness in the new population. After a terminating at a number of generations in this case, we took from the final population a string obtained as an average of the top 10 "best" strings, rounding to integer values in both components of the string after averaging. This string was taken to be the final knowledge base obtained by the algorithm. It has been found to be a very successful operation as all the knowledge in this process appears very difficult to achieve in an single best performing string. Finally to measure the performance of this final set of fuzzy rules its objective value was evaluated at some 20 randomly selected different configurations in the grid.

These modifications clearly increase quite drastically the amount of computation and execution time. This consideration is traded off against the successful computation of the N knowledge bases and the averaging required in (i). The grid was taken uniform with maximum $|x_1| = 0.75$ and maximum $|x_3| = \pi/12$ when $N = 256$. Results so far obtained for the single knowledge base with initial selection of ω_k given above show slow convergence and after 1,900 generations on average 2/3 terminal constraints being broken. For the two knowledge base case with a smaller grid $N = 50$ and maximum $|x_1| = 0.35$, after 1000 generations on average no terminal constraints are being broken.

6 Conclusion

We have determined using an evolutionary learning algorithm fuzzy rules for a single and two layer knowledge base for the control of the inverted pendulum. It is fast learning in general from a given initial configuration. The algorithm has been extended to find knowledge bases that will effectively control the system within specified bounds from a wide range of initial configurations. The research is current. The two knowledge bases determined had only 13 rules in common with those developed experimentally for the pole-cart system constructed at the Singapore Polytechnique. The given nonlinear state equations do not represent a truly accurate model of their system. Nevertheless it is worthwhile to compare their fuzzy rules with those that we have obtained via the evolutionary learning algorithm.

References

[1] J.J.E. Slotine and W. Li, *Applied Nonlinear Control*, Prentice Hall, 1991.

[2] C.T. Lin and C.S.G. Lee, *Neural Fuzzy Systems*, Prentice Hall, 1996.

[3] M.O. Odetayo and D.R. McGregor, Genetic Algorithm for Inducing control Rules for a Dynamic System, Proc. of the 3rd Int. Conf. on Genetic Algorithms, George Mason University, pp. 177-182, 1989.

[4] A. Varsek, T. Urbancic and B. Filipic, Genetic Algorithms in Controller Design and Tuning, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 23, No. 5, pp. 1330-1339, 1993.

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

[6] M.A. Lee and H. Takagi, Integrating design stages of fuzzy systems using genetic algorithms, Proc. IEEE Int. Conf. Fuzzy Systems, Vol. I, pp. 612-617, San Francisco, 1993.

[7] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.

[8] R.J. Stonier and M. Mohammadian, Knowledge Acquisition for Target Capture, Proc. of the IEEE Int. Conf. on Evolutionary Computing May 1998, Anchorage, Alaska.

[9] R.J. Stonier and M. Mohammadian, Self Learning Hierarchical Fuzzy Logic Controller in Multi-Robot Systems, Proc. of the IEA Conf. Control95, Melbourne Australia, pp. 381–386, October 1995.

[10] R.J. Stonier and M. Mohammadian, *Evolutionary Learning in Fuzzy Logic Control Systems, Complex Systems 96; From Local Interactions to Global Phenoma*, Eds. R.Stocker et al., IOS Press, pp. 193–212, July 1996.