Learning Fuzzy Control by Evolutionary and Advantage Reinforcements

Munir-ul M. Chowdhury and Yun Li

Centre for Systems and Control, and Department of Electronics and Electrical Engineering, University of Glasgow, Rankine Building, Glasgow G12 8LT, UK

Abstract: In this paper, evolutionary and dynamic programming based reinforcement learning techniques are combined to form an unsupervised learning scheme for designing autonomous optimal fuzzy logic control systems. A messy genetic algorithm, and an advantage learning scheme are first compared as reinforcement learning paradigms. The messy genetic algorithm enables flexible coding of a fuzzy structure for global optimisation, resulting in a coarsely optimised feedforward type neurofuzzy structure. Local pruning and fine tuning of the neurofuzzy system is then achieved effectively by advantage learning by directly interacting with the environment without the use of a supervisor. The methodology is illustrated and tested in detail through application to two nonlinear control systems.

Keywords: Neurofuzzy Control, Messy Genetic Algorithms, Reinforcement Learning

1. Introduction

It is known that fuzzy logic control (FLC) systems may be used as an alternative to some conventional control schemes where significantly improved system behaviour and user-friendly decision making mechanism can be obtained. However, there exists no guidelines for designing fuzzy controllers. Manual designs may require long period of trial and error and much input from experts are required, as it is difficult to defend the choice of, for example, any particular type of membership functions or the reasoning structure. What may be adequate for one set of conditions, may not be appropriate under similar but different conditions, as fuzzy control systems lack learning ability. To improve the performance of a fuzzy control system, the parameters in terms of structure, complexity, number of neurons, and type of neurons etc have to be tuned by an optimisation method. In recent years, it has become clear that neural and fuzzy hybrid systems have advantages such as the applicability of existing algorithms for artificial neural networks (ANNs), local controller structures and direct adaptation of knowledge expressed as a set of fuzzy linguistic rules. A difficulty facing various existing neurofuzzy hybrids is that learning is supervised and thus requires training information on the subject domain. In addition the network structures are prefixed and may only be appropriate for a limited set of problems and the learning algorithm may be trapped in local optima. Obtaining effective and quality training data for engineering systems is probably the primary difficulty because often it is not possible to do so or how to produce the training data is not in general predictive and cannot truly represent the varying environment.
One method used in solving this type of problems is based on an unsupervised learning paradigm known as reinforcement learning (RL). The learning technique is often used for training intelligent (neural) controllers in a supervised/unsupervised mode where only minimal a-priori knowledge is available. Due to the limited amount of available information, the controller has to learn an appropriate action that transfers an unknown system from its current state to a target state, which is expected to be superior.

In the work reported in this paper, two forms of reinforcement learning are applied, the first being based on an evolutionary algorithm and the second on the dynamic programming approach. For the evolutionary approach a messy genetic algorithm (mGA) is used. The primary characteristic of this algorithm is the ability to represent information using a variable chromosome size. This enables flexible encoding of the network properties such as connectivity, number of neurons and their types. This approach tends to optimise the controller globally, but may not arrive at the exact optimum, i.e., the design may be coarse. For the dynamic programming approach, advantage learning is used. Advantage learning learns a function of state/action pairs where the value associated with each action is called an advantage. The value of the state is defined to be the value of the maximum advantage in that state. This second approach is used to fine-tune the near optimal or coarsely designed controller further at a higher speed by pruning the parameters of the neurofuzzy network.

In section 2, the development of neurofuzzy networks is described to underline the scene. Section 3 compares and summarises the two reinforcement learning approaches. Their unification and application to neurofuzzy control are detailed in Section 4. Section 5 demonstrates the methodology through the control of two nonlinear systems. Conclusions are highlighted in Section 6.

2. Neurofuzzy Control

Neural network and fuzzy logic theories were developed about the same period of time. Recently, a main direction of research combining these two has been towards automatic design and fine-tuning of the membership functions used in fuzzy control through learning by artificial neural networks. ANNs are of a massive parallel structure of highly nonlinear processing elements, whose weights and characteristics may be “trained”. Fuzzy systems are also of a parallel structure but are more suitable for knowledge extraction and representation. On the other hand, the weak points of fuzzy systems are the difficulty of defining accurate membership functions and of applying the learning method. One of the most obvious similarities between a fuzzy system and an ANN is that they can both handle extreme nonlinearities in the system collectively by a network of “local” elements such as memberships or neurons. The functionality of the position of the membership function in the fuzzy system and that of the threshold function in the ANN, are similar. The multiplication-addition operation of artificial
neurons is very close to the **MAX-MIN** operation of approximate reasoning. The **MIN** operation of input fuzzy variables conducted at each proposition of **IF** part of a fuzzy inference rule corresponds to the product of an input and its weight. The **MAX** operation to obtain the final inference value from a **THEN** part of a plural inference rule corresponds to the sum of the weighted input. These reasons lead to the idea of merging these two paradigms.

The following is a summary of the main results, drawn from literature, regarding computational equivalence between a neural network and a fuzzy system\(^1\,{}^5\,{}^7\)\(^9\):

1. A Feedforward neural network with \(n\) inputs, \(m\) outputs (\(n \geq 1, m \geq 1\)), one or more hidden layers, and a continuous activation function (e.g., the sigmoid function) in each neuron can be a universal functional approximator.
2. A Fuzzy system based on multiconditional approximate reasoning can approximate a feedforward neural network with \(n\) inputs, \(m\) outputs, one or more hidden layers, and a continuous activation function in each neuron, provided that the range of the input variable is discretised into \(n\) values and the range of the output variable is discretised into \(m\) values.
3. It follows from (1) and (2) that fuzzy expert systems of the type described in (2) is also a universal approximators.
4. A Fuzzy input-output controller based on multiconditional approximate reasoning and a defuzzification of obtained conclusion, is also a universal approximator.

Early attempts at combining neural networks and fuzzy control were limited to just tuning the shapes of the membership functions. Such work was reported by Nomura et al., where the membership functions are assumed to be symmetrical triangular functions depending on two parameters, the peak and the width\(^7\). Fuzzy cognitive maps proposed by Kosko is another scheme to integrate neural networks and fuzzy logic\(^1\). Here, the membership function or fuzzy rules are chosen subjectively. Lin proposed a general feedforward multilayer neural network for fuzzy logic control and decision systems\(^8\). The fuzzy logic components are directly integrated in the neural network. The input and output nodes represent the input states and control signals, respectively, and in the hidden layers there are nodes that code membership functions and rules. The learning algorithm used for building rule nodes and training the membership functions is based on the backpropagation algorithm.

What is evident from all these proven structures is that in all of them the following features are common:

1. Inputs are real numbers which are fuzzyfied by the neurofuzzy structure.
2. Outputs are fuzzy numbers.
3. Weights are generally fuzzy numbers.
4. Weighted inputs of each neuron are not aggregated by summation, but by some other aggregation operation as defined by the fuzzy inferencing.
The neurofuzzy controller structure developed in this paper also exhibits these properties, and is a concatenation of two fuzzy networks, as depicted in Fig 1. The first network represents the premise of the equivalent fuzzy system, layer 1 and 2, and the second the consequence, layers 3 to 5. The inputs to the second network is the normalised output of the first network. Since the network essentially represents a fuzzy logic controller mapping there are restrictions on how much the network can be adjusted in order to achieve the desired actions from the systems, e.g. the number of layers cannot be altered since this has a direct relation to the inferencing mechanism. This limits the structural optimisation to the type of activation function of the neurons; the number of neurons per layer and the necessary links between adjacent layers. Therefore, the relevant parts of the network requiring optimisation are layers 1, 2 and 4 as only these influence the action of the controller. The other parts on the network are kept constant.

The NFC structure resembles a feedforward network. The output of every node of layer 1 is simply the grade of membership, $\mu_i(x_i)$, as a consequence of mapping the crisp input value onto its corresponding membership function. Every node in this layer is a fixed node labelled $\land$, and essentially performs the AND operation of the fuzzy inferencing. Each node output represents the firing strength of a rule. The output of each node is defined by

$$w_{ji} = \mu_{a_i}(x_i) \times \mu_{b_i}(x_2), \quad i = 1,2$$  (1)
Layer three performs the normalising operation. The $i$-th node calculates the ratio of the $i$-th rule’s firing strength to the sum of all rules’ firing strengths:

$$w_{2i} = \frac{W_{hi}}{\sum_{w}^{}}, \quad i = 1, 2$$

The output of nodes of layer 4 is based on the defuzzification process. For a Sugeno type defuzzification where the output membership is simply represented by spikes (or singletons) the node function is given by:

$$w_{3i} = w_{2i}\left(p_ix_i + q_ix_i + r_i\right)$$

where the triplet $\{p_i, q_i, r_i\}$ is the parameter set. Finally crisp output is obtained by computing the overall output as the summation of all incoming signals. The error function employed can, for example, be a relative entropy function defined by:

$$E = \sum_{i} \left[\frac{1}{2}\left(1 + \zeta_i\right)\log\frac{1 + \zeta_i}{1 + y_i} + \frac{1}{2}\left(1 - \zeta_i\right)\log\frac{1 + \zeta_i}{1 - y_i}\right]$$

where $\zeta_i$ is the desired output and $y_i$ the actual. The Entropy function has an advantage over the standard quadratic error function in that it accelerates convergence on plains in the error information landscape where the standard function could stick and decelerates progress on sharp bends of the cost surface.

### 3. Reinforcement Learning Schemes

The main problem with existing neurofuzzy hybrids is that they are based on supervised learning algorithms. This implies that sample input-output pairs for the function to be learned has to exist. Unfortunately, there are many situations where we do not know the correct answers that supervised learning requires, and where the learning has to be adaptive to the environment. For example, in a flight control system, the question would be the set of all sensor readings at a given time, and the answer would be how the flight control surfaces should move during the next millisecond. For these reasons there has been much interest recently in a different approach known as reinforcement learning. Reinforcement works by directly interacting with the environment, and the performance of the learning agent is based on a single reinforcement signal without indicating the direction in which the system could be improved.
In application to the optimal control problems RL could be formulated in a way that the long-term consequences of actions are taken into account since in most of the cases the goal is to design a controller with an optimal long-term performance. The RL based controller is then designed to receive a reinforcement signal from the controlled process based on its performed action and the state of the process. The objective of the learning system is then to either minimise or maximise the amount of reinforcement signals accumulated in the future, depending on what the signal represents, cost or benefit. This performance measure is often calculated as a discounted sum of the future reinforcement signals in which the earlier ones are weighed more.

In a reinforcement learning model an agent interacts with the learning environment. This interaction takes the form of the agent sensing the environment and, based on this sensory input, choosing an action to perform in the environment. The action changes the environment in some manner and this change is communicated to the agent through a scalar reinforcement signal. The reinforcement learning problem is composed of three components: the environment, the reinforcement function, and the value function. The RL system interacts with the environment to obtain a mapping from situation to actions by trial and error. The environment provides the RL system information in the form of sensor readings, and based on this information the RL system chooses actions based on the states of the environment. The exact function of future reinforcements the agent tries to maximise is the reinforcement function. The RL agent learns to perform actions to maximise the sum of the reinforcement signal received going from an initial state to a terminal state. Finally the purpose of the value function is to map state to state values using some sort of function approximator such as fuzzy rulebases. In this work, two types of reinforcement learning systems are considered based on genetic algorithms and dynamic programming.

3.1 Evolutionary Reinforcement

Traditional learning and optimisation methods work around a single point by a priori and prone to problems such as getting trapped in a local optimum. Genetic algorithms (GAs) are global optimisation procedures based on the processes that appear to be at work in biological evolution and the working of the immune systems. Coded information is presented in genes which make up chromosomes, and genetic algorithms operate on structures similar to these chromosomes. In contrast to conventional optimisation methods such as gradient descent, a GA varies system parameters instead of the parameters themselves using operators called crossover and mutation. It then guides the selection of fit chromosomes from the new group and home the search a posteriori. GAs in contrast are non-deterministic, do not require derivative information and work with a population of points making it much more robust as it is more likely to lead to global optima. In the context of reinforcement learning, policies are learnt directly. Through crossover and mutation, new pool of policies are evolved. The credit assignment is carried out
by submitting individual structures to evaluation, and assigning new strengths using an objective or reinforcement function. The relative performance of different chromosomes representing different solutions is used by the agent to select the best action (population member) for future evaluations until terminal conditions are reached.

In a “regular” GA, a coded chromosome is in fixed length that highly fit allele combinations are formed to obtain a convergence towards global optima. Unfortunately the required linkage format (or the structure of the controller to be coded) is not exactly known and the chance of obtaining such a linkage in a random generation of coded string is poor. Poor linkage also means that the probability of disruption on the building block by the genetic operators is much higher. Although inversion and reordering methods can be used to adaptively search tight gene ordering, these are too slow or effective to be considered useful.

3.2 Messy Genetic Algorithms

A type of GAs, the messy genetic algorithm exists which uses variable length chromosomes. The algorithm is summarised in Fig 2. The main difference between an mGA and a regular GA is that

- The mGA uses varying chromosome lengths.
- The coding scheme considers both the gene index and values.
- The crossover operator is replaced by two new operators called cut and splice.
- It works in two phases - primordial phase and juxtapositional phase.

We start by defining a template which is used to fill in any unnamed genes in a chromosome. If one has some knowledge of the structure of the controller then this template can be an initial solution. Alternatively the template can be initialised randomly or set to zero and built up with each generation. The template is updated after each generation using the best solution of that generation. Instead of entering a generational loop as in the traditional GA, mGA is executed in two phases, primordial and juxtapositional.

The primordial phase is concerned with enriching the population with optimal or near optimal chromosomes. No genetic operators are applied, only the selection mechanism is used. The population is first initialised to contain all possible chromosome lengths of a specified length. At the end of specified periods the population size is halved, keeping the best solutions. The juxtapositional phase follows the primordial with a fixed population size, and the generational loop of Fig 2 is followed, the only difference being that the crossover operator is replaced by two new ones, i.e. cut and splice as shown in Fig 3. The cut operator severs a string with specified probability \( p_c = (\lambda - 1)p_c \) that grows as the string length \( \lambda \), and splice joins two strings together with fixed probability \( p_s \).
It can be observed from Fig 3 we observe that the genes do not appear in any specific order. In fact the order of the genes are irrelevant in an mGA. Another characteristic of the mGA is that as a result of the variable chromosome structure, the chromosome may be underspecified or overspecified. Underspecification is the case when certain genes are missing from a specific chromosome. If the chromosomes in Fig 4 represented a four parameter problem, then parent 2 before cut and splice is underspecified because there is no reference to parameter (gene) 2. This is when the template is applied, and in this case the value for parameter 2 is extracted from the template. We also observe that in parent 1 of Fig 4, reference to parameter (gene) 1 is made more than once. This is termed overspecification and some sort of precedence rule, such as first come first served, is applied to handle chromosomes of this type.
3.3 Advantage Learning

Advantage learning is another reinforcement learning paradigm developed by Marmon and Baird\(^6\). It can be regarded of as an extension of the family of reinforcement learning algorithm known as Q-learning\(^{12}\). In Q-learning there is a \(Q\)-value associated with each action which is the sum of the (possibly discounted) reinforcements received when performing the associated action and then following the given policy thereafter. The optimal \(Q\)-value is defined as the sum of the reinforcements received when performing the associated action and then following the optimal policy thereafter. In the context of Q-learning, the value of a state is defined to be the maximum \(Q\)-value in the given state. Given this definition we can now define derive the function for Q-learning as:

\[
Q^\pi(x_t, u_t) = r(x_t, u_t) + \gamma J^\pi(y_t)
\]

where \(y\) is the state resulting from applying action \(u(x)\) to state \(x\), \(\gamma\) is the discount factor, and \(r\) is the reinforcement signal at time \(t\). Assuming \(Q^*\) represents the optimal \(Q\)-function, then the optimal value function

\[
J^*(x_t) = \max_u Q^*(x_t, u_t)
\]

This leads us to define the optimal Q-learning rule as

\[
\hat{Q}(x_t, u_t) = \hat{Q}(x_t, u_t) + \alpha \left[ r(x_t, u_t) + \gamma \max_{b_t} \hat{Q}(y_t, b_t) - \hat{Q}(x_t, u_t) \right]
\]

The advantage of the Q-Learning is that once the Q values are learned, then one need only find the maximum \(Q\)-value in the new state to have all the necessary information for revising the
prediction ($Q$-value) associated with the action just performed. $Q$-learning does not require one to calculate the integral over all possible successor states in the case that the state transitions are non-deterministic. One major drawback to the Q-learning algorithm is that the number of training iterations required to adequately represent the optimal Q-function scales poorly with the size of the time interval between states. This means that the smaller the time step between successive actions, the greater the number of training iterations are required to adequately represent the optimal Q-function. To overcome this, Marmon and Baird proposed the Advantage Learning or A-learning paradigm which avoids this scaling problem. Like Q-learning, A-learning also learns a function of {state, action} pair, where the value associated with each action is called an advantage. The value of the state is defined to be the value of the maximum advantage in that state. The advantage is defined to be the sum of the value of the state and the advantage of performing action $u$ instead of the best current action. Advantage Learning rule is given by

$$A(x,s_t) = \max_{u_t} A(x,s_t) + \Delta T \left( \mathbb{E} \left[ r(x_t, u_t) + \gamma \max_{u_t} A(x_{t+1}, u_t) \right] - \max_{u_t} A(x_t, u_t) \right)$$

where $\Delta T$ is a time unit scaling factor.

4. Unified Reinforcement Neurofuzzy Scheme

The use of genetic algorithms for optimising fuzzy and neurofuzzy structures have been in existence for some time, and these are broadly based on model based and supervised algorithms. In addition, the flexibility of the structure is compromised between complexity of the problem and computational effort. For example, it may be necessary to represent a variable with 9 fuzzy sets. If one considers the general format for representing fuzzy sets within the chromosome, with each gene representing a parameter of the fuzzy set (such as the spread and centre for gaussian sets) then there would be need for 18 (9*2) genes to represent that variable. This is obviously not very desirable if there are a large number of such variable to optimise. In addition of not knowing how many sets to represent each variable with, one also has to consider the type of the sets and the form of the rules defining the {state, action} pair. In addition, any redundant information identified as a result of the evolution process is not removed from the process as it still remains in the coded structure. For these reasons, the flexible messy GA is used.

More recently there have been numerous attempts to apply dynamic programming RL methods to fuzzy systems. The majority of these being based on Q-learning and applied to classifier system where patterns are matched using fuzzy linguistic type if-then rules. Usually, in
an FLC, some rules triggers on the same crisply defined state, and together co-operate to produce an action. There is a one-to-one mapping between an agent (i.e. a set of rules) and the action it produces. Therefore, the performance of each agent is evaluated independently from that of each other. The difficulty with this method is that the rule structure contains all possible combination of rules making it computationally inefficient. The Berenji method uses fuzzy constraints, instead of fuzzy rules, among the actions that can be done in a given state\textsuperscript{18}. At each step, the action with the maximum Q value is selected, and brings the system in a new state. In this case, actions are not combined, as in traditional fuzzy systems, but only selected. The Bonarini method is a fuzzy classifier system based on the ideas of GAs\textsuperscript{25}. It works on a population of fuzzy rules, and associates each rules with its strength and its contribution to past actions. The population of rules are allowed to have the same premise for different consequents, thus introducing an element of competition within the process to find the best consequence for any given premise state. Another method to use the evolutionary based reinforcement learning was proposed by Whitley\textsuperscript{21}. In this method the system receives a signal of success or failure from the real world, and the system learns from the strength of this signal to improve its success rate. However, since the quality of such feedback signal is generally poor, learning is inefficient. Another drawback is noisy fitness function and biased signals which meant that not all possible state occurrences were learned.

In Fig 1 we observe that the network is essentially a mapping of the various fuzzy stages, and the optimisation of the network is limited to the type of activation function of the neurons; the number of neurons per layer and the necessary links between adjacent layers. Therefore, the relevant parts of the network requiring optimisation are layers 1, 2 and 4 as only these influence the action of the controller. Having obtained a general structure of the network which gives a near optima solution, fine tuning of the network parameters such as fuzzy set parameters and connectivity (rule structure) is carried out using the Advantage learning algorithm.

The first stage therefore deals with obtaining the structure of the network using the mGA procedure. The regions of fuzzy subspaces are defined according to the information available about the plant to the operator. Where the operating regions are known, a fixed universe of discourse with varying size membership functions. When the operating region range is not so clear, fixed size membership functions with a varying universe of discourse is used. The resulting network is one where the entire operating region is well covered with equally spaced overlapping membership functions, enabling smooth transition between states.

When optimising using mGA, each gene is a set of numbers that indicates the I/O index, the neuron of the adjacent layer it connects to and the type of activation of the neuron (fuzzy set shape). Using the mechanism of the mGA, a candidate neurofuzzy controller system may be coded as a string shown in Fig. 4. Assuming there are 2 inputs and one output to the system, then the string \(((1,1,2) (3,2,2) (3,0,1) (2,3,1))\) would be interpreted as:
(1,1,2): Input 1 connects to the 1st neuron of layer 2 which has activation of type 2.

(2,3,1): Input 2 connects to the 3rd neuron of layer 2 which has activation of type 1.

(3,2,2): Output 1 connects to the 2nd neuron of layer 4 which has activation of type 2

![Diagram of mGA gene coding](image)

Fig. 4. Coding of mGA gene

To efficiently code the genes, an enhanced scheme based on Ng\textsuperscript{26} can be employed. First a decimal mapping is obtained from an integer string:

\[ C = C_{\text{min}} + \frac{a_{p-1}b^{p-1} + \ldots + a_0b^0}{b^p}(C_{\text{max}} - C_{\text{min}}) \]  \hspace{1cm} (9)

where \( C \in [C_{\text{max}} - C_{\text{min}}] \) is the decimal value being coded, \([C_{\text{max}}, C_{\text{min}}]\) denotes the decoding range, \( b \) is the base value for coding, \( a_p \in [0, b-1] \) is an unsigned integer code and \( p \) is the number of digits used in the decoding which indicate the compromise between accuracy and speed in the evolution process. After which the decimal value is once again manipulated to give information about the rule structure. For example a decimal value 39.2 would be broken up into a vector \( [3 9 2] \) when the first column indicates the input-output variable. The second column refers to the node of the adjacent layer to which the I/O variable connects to and the last column refers to the activation type of that node. If the node is assigned an activation from a previous decoding operation, then that activation is used.

On completion of the first stage, the best network structure is passed to the second stage where pruning and fine-tuning of the network is carried out by A-learning. Random initial states are applied to the network which is then allowed to run until the system fails. If running off-line, new states to the controller is obtained as before using the RK algorithm. The number of successful \( \{x, u(x)\} \) (i.e. \( \{\text{state, action}\} \)) pairs are recorded and used as the overall reinforcement
signal. The weights of the network are updated according to the number of hits they receive during each cycle of the learning algorithm.

If certain weights (corresponding to the parameters of the membership function (MF)) are used more often than others then they are rewarded such that the membership function’s base width is increased and it’s adjacent MFs are penalised by being reduced.

Consider the equivalent fuzzy sets represented by the neurons at the input and final output layers of Fig 1. Let the shapes of these fuzzy sets be of triangular shape defined as

\[
\text{Triangle}(x; a, b, c) = \max \left( \min \left( \frac{x-a}{b-a}, \frac{c-x}{c-b} \right), 0 \right)
\]

(10)

where \(a, b, c\) are parameters of the set as shown in Fig 5. We define \(x(C_{s_1}, C_{s_r})\) and \(u(C_{s_1}, C_{s_r})\) as the [state, action] pair, and \(C_{s_1}, C_{s_r}\) are the supports of the fuzzy set.

![Fig 5 Triangular activation function](image)

\[
C_{s_1}(t) = \Delta S_{s_1}(t)C_{s_1}(t-1)
\]

(11)

\[
C_{s_r}(t) = \Delta S_{s_r}(t)C_{s_r}(t-1)
\]

(12)

where \(\Delta S\) is the amount the support is shifted,

\[
\Delta S = \frac{R'}{R} N
\]

(13)

where \(R'\) is the number of unique rules exciting set, \(R\) is the total number of rules as found at stage one of the process and \(N\) is a weighting factor given by

\[
N = \frac{\mu_s(m)}{\max(\mu_s(m))}
\]

(14)
where $s$ is the set under consideration and $S$ is the full range of sets for the state and $m$ is the linguistic rule. Finally to use the advantage learning of (8) we define the reinforcement signal as

$$
 r_t = \sum_k \gamma^{k-1} r_{t+k} + \gamma^n f_i(x_{t+n})
$$

(15)

where $f$ is the objective or value function and $\gamma (0.95)$ is the discount factor. The algorithm is illustrated in Fig 6. Consider an initial weight set-up such that the corresponding memberships as in Fig 6a. Now assuming weights corresponding to the second membership function “NS” receive the most attention, then the base of this MF is spread out more into the regions of its adjacent MFs - “ZE” and “NM”, Fig 6b At the same time, as a consequence, the base of its adjacent MFs are also reduced because they are playing a smaller role in the process. The amount of increase and decrease is proportional to the amount the weights are activated.

This type of reward-penalty policy also has the advantage of removing redundant MFs. Consider Fig 6a again. In addition to “NS” receiving the most attention, assume “PS” is receiving more attention than “ZE” and “PM”, then the base of “PS” would also be increased and the base of “ZE” and “PM” reduced. So we can see “ZE” being gradually squeezed out from both sides. If as a result of increasing both “NS” and “PS”, “ZE” is completely encompassed or not activated at all then “ZE” is removed. The cycle of this second stage is repeated until the plant operates successfully to the operators satisfaction (such as for a specified period). In summary, the overall process is described in Algorithm 1.

**Algorithm 1** Evolutionary RL of neurofuzzy networks

**Stage 1: Off-line structural optimisation**

- Identify inputs and outputs
- Identify approximate fuzzy subspaces for inputs variables
Obtain an approximate fuzzy rule base and network mapping
Optimise fuzzy subspaces and structure of the network using mGA

Stage 2: On-line network weight tuning
Start with the off-line network to obtain controlled actions
While system is successful within limits apply new input states to the network
Record time to failure to reward and reinforce the best networks using Advantage Learning

5. Applications to Nonlinear Control

5.1 A Single-Output Asymmetric Nonlinear System

We now apply the methodology as sketched in to an asymmetric nonlinear laboratory coupled liquid-level regulation system, Fig 7, to demonstrate and validate the method. The system is analogous to plants widely involved with chemical and other processes involving mass balancing or heat-balancing. The system description is given by the state-space differential equation set:

\[
\begin{bmatrix}
\dot{h}_1 \\
\dot{h}_2
\end{bmatrix} = \begin{bmatrix}
\frac{-C_1 a_1}{A} \sqrt{2g|h_1 - h_2|} \\
\frac{C_1 a_1}{A} \sqrt{2g|h_1 - h_2|} - \frac{C_2 a_2}{A} \sqrt{2g|h_2 - h_0|} + \begin{bmatrix}
\frac{1}{A} & 0 \\
0 & 1
\end{bmatrix}\begin{bmatrix}
u \\
d
\end{bmatrix}
\end{bmatrix}
\tag{16}
\]

Here, \(h_1(t)\) and \(h_2(t)\) are the liquid levels of Tank 1 and Tank 2, respectively; \(u(t)\) is an input flow rate mapped from a pump voltage; \(d(t)\) is also a pumped input but is used to test the rejection of disturbances when needed; \(C_1 = C_2 = 0.58\) are discharge constants; \(a_1 = 39.56 \times 10^{-6} \text{ m}^2\) and \(a_2 = 38.5 \times 10^{-6} \text{ m}^2\) are orifice areas; \(A = 0.01 \text{ m}^2\) is the cross-sectional area of both tanks; \(h_0 = 0.03 \text{ m}\) is the minimum liquid level bounded by the height of the orifices, and \(g = 9.81 \text{ m s}^{-2}\) is the gravitational constant.
The objective of this control system is to drive, through the input to Tank 1, the liquid level at Tank 2 towards the desired level of 0.1 m in the first control cycle as fast as possible with minimal overshoots and steady-state errors. A second control cycle takes place from 600 s and the desired level in Tank 2 now is 0.2 m.

The input states to the neurofuzzy network are the tank level, rate of change of the level. The output is the pump flow rate. For stage 1, the mGA was configured to accommodate a maximum of nine membership functions for each state variable. Since there are obvious limitations to the amount the liquid level can rise and capacity of the pump, a fixed universe of discourse scheme was used. The activation type used was the three parameter triangular form of (10). The evolution of this nonlinear system is slow and hence to keep the computational time to a minimal possible only one type of activation is used. We will also see that the triangular shape gives us greater freedom when we attempt to fine tune the network in Stage 2. The initial template was defined as a fully connected network with 9 memberships to describe each state/action variables. An initial population size of 300 was used to enable a faster convergence. This was halved after each era for 2 eras and left a population size of 75 after the primordial phase. For the juxtapositional phase the cut and splice rates were set to 75% and 80% respectively. Mutation rate was set to 10% and genes were mutated to a value not in the chromosome. The mGA evolution progress curve for 100 generations is shown in Fig 8. The resultant network of the system after stage 1 is illustrated in Fig 9 and the response of the closed loop system is given in Fig 10. We see that the response is reasonable but not as smooth as one would desire. There appears to be excessive switching effect taking place and so much tuning is required.

The resulting “best” network is carried forward to the second stage for local learning and fine tuning. For each training state if the network was able to deliver a level of liquid in tank 2 to within 5% of the desired level, it is rewarded with a score of 1 and –1 if outside the 5% boundary. With the objective not only to reach the desired height in the tank but also to maintain the level for a set period of time, the cumulative score after each run is used to reinforce the “local learning”. The process is repeated until a satisfactory response is observed.
Fig 11 shows the resulting network of the system after local learning and Fig 12 shows the response. We note that there is no steady state errors and no switching effect. Comparing with Fig 9, we observe that in addition to the shape of the activations being affected, the number of neurons and the network connectivity is simplified. To test the controllers robustness and ability to generalise in dealing with a nonlinear system with varied operating conditions, the resultant controller was tested for different desired levels with a constant disturbance inflow of $8.33 \times 10^{-5}$ m$^3$/sec at an interval of every 300 s. Fig 13 shows the responses for this test. We observe that the disturbances are almost rejected and the tank level is within “acceptable” limits.

![Fig 8  mGA Evolution Progress](image)

![Fig 9a  Neurofuzzy network for tank system after stage 1](image)
Fig 9b  Extracted fuzzy sets from neurofuzzy network after stage 1

Fig 10  Closed loop response of tank control system after stage 1
Fig 11a  Neurofuzzy network of tank system after stage 2

Fig 11b  Extracted fuzzy sets from network of tank system after stage 2
Fig 12  Closed Loop response of tank system after stage 2

Fig 13  Closed loop response with disturbance
5.2. A Multi-Output Unstable Nonlinear System

The second example is a highly nonlinear cart-pendulum system. However, instead of using the traditional single pendulum we use the more complex double pendulum system as shown in Fig 14. This non-linear control problem is selected because of its similarity to many practical engineering applications, such as robot balancing, space shuttle arm, ballistics, and mathematically similar factory roof cranes, which require precision, stability and flexibility. The objective is to, by applying a control force to a cart centre of mass, centre the cart on the track and balance both pendulums to vertical axis. Using a Lagrange approach, the dynamic equations of the system are found as

\[ h_1 \ddot{x} + h_2 \ddot{\alpha}_1 \cos \alpha_1 + h_3 \ddot{\alpha}_2 \cos \alpha_2 - h_2 \ddot{\alpha}_1 \sin \alpha_1 - h_3 \ddot{\alpha}_2 \sin \alpha_2 = u \]  

(17)

\[ h_2 \ddot{\alpha}_1 \cos \alpha_1 + h_4 \ddot{\alpha}_1 \cos (\alpha_1 - \alpha_2) - h_3 \ddot{\alpha}_2 \sin (\alpha_1 - \alpha_2) - h_5 \sin \alpha_1 = 0 \]  

(18)

\[ h_3 \ddot{\alpha}_2 \cos \alpha_2 + h_6 \ddot{\alpha}_2 \cos (\alpha_1 - \alpha_2) + h_5 \ddot{\alpha}_1 \sin (\alpha_1 - \alpha_2) - h_8 \sin \alpha_2 = 0 \]  

(19)

where \( h_1, \ldots, h_8 \) are defined as below:

\[ h_1 = m_c + m_1 + m_2 \]
\[ h_2 = m_1 L_1 + m_2 L_2 \]
\[ h_3 = m_2 L_2 \]
\[ h_4 = m_1 L_1^2 + m_2 L_2^2 + J_1 \]
\[ h_5 = m_2 L_2 L_1 \]
\[ h_6 = m_2 L_2^2 + J_2 \]
\[ h_7 = m_1 L_1 g + m_2 L_2 g \]
\[ h_8 = m_2 L_2 g \]

Here \( x \) is the cart position, \( \alpha_1 \) and \( \alpha_2 \) are pendulum link angles in radians, \( L_1 \) and \( L_2 \) are lengths of pendulum links, \( g = 9.81 \text{ms}^{-2} \) is the gravitational constant, \( l_1 \) and \( l_2 \) are the distances between the pivot and centre of mass of respective links, \( u \) is the control force, \( G_1 \) and \( G_2 \) are the centre of mass of primary link second link respectively, \( m_c \) (= 1.5 kg) is the mass of cart, \( m_1 \) (= 0.5 kg) and \( m_2 \) (= 0.75 kg) are the masses of first and second links and \( J_1 \) and \( J_2 \) (=0.0005kgm^2) are the inertia of first and second links about their centre of mass.

The inputs to the neurofuzzy network are cart position, cart velocity, the pendulum link angles and their angular velocities and the output variable is the control force. As with the previous example, For Stage 1, the mGA was configured to accommodate a maximum of nine membership functions for each state variable. This time the only bounds being the length of the track. Ideally we would like the pendulum to be able to operate successfully from any given
angle, but in practice this is not possible, hence again a fixed universe of discourse scheme was used. The activation type used was the triangular and gaussian bell shaped form.

Since there are 6 inputs, if we try to use a single controller in simple fuzzy form we would need a 6-D rule base which would be too complex to implement. However, this does not impose a significant problem to the neurofuzzy structure. Once again, the initial template was defined as a fully connected network with 9 memberships to describe each state/action variables. For this problem an initial population size of 200 was used because it is a much faster system. Again the population was halved after each era for 2 eras. This left a population size of 50 after the primordial phase. For the juxtapositional phase the cut and splice rates were set to 80% and 80% respectively. Mutation rate was set to 15% and genes were mutated to a value not in the chromosome.

For the second stage, scores were awarded if the cart was within the track limits, and the pendulums did not fail. Once again an error margin of 5% was defined as “satisfactory”. Fig 17-18 illustrates the behaviour of the cart-pendulum system with the optimised controller. We observe all the states reaching the desired states to a fast settling time and little oscillation. Only a snap shot of the entire time process is displayed to show the behaviour of the pendulums and cart before reaching goal state.

Fig 14 Cart-double pendulum co-ordinate system
Fig 15  Cart Position and Velocity

Fig 16  Angle dynamics of pendulum 1
6. Discussion and Conclusions

A simple neural network like structure can be used to map a fuzzy system. This enables the controller to exhibit both inferencing and learning properties. In this paper, a design process has been devised which utilises evolutionary, reinforcement and neural paradigms to implement and enhance these properties. The evolutionary algorithm based flexible optimisation scheme has been combined with neurofuzzy systems to obtain controllers providing globally searched optimal performance. The algorithm uses efficient coding and representation. The advantage of using GAs as an RL tool is that it is based on being responsive to changes in the environment.

However, although the GA approach tends to optimise the controller globally, it leaves the design coarse. In addition, the evolutionary approach requires that the total reward be available before the information can be used. Local learning is carried out using the Advantage learning
algorithm. This approach makes better use of computing resources because total reward in not necessary to define a partial success, high intermediate state is sufficient. The advantage of this unified approach, as opposed to using alternative methods such as evolutionary programming and evolution strategy algorithms, is that it allows a neural network topology to be coded and optimised at the same time as weights training. In particular, the mGA algorithm applied here permits architectural flexibility in the neurofuzzy network design and in fuzzy logic system optimisation. The resulting networks tend to be globally efficient and locally refined.

This approach has been applied to two nonlinear control problems which demonstrated the simplicity in application and validated the methods.

Acknowledgements
The first author wishes to acknowledge the funding by the Engineering and Physical Sciences Research Council of the UK for assisting in the funding in the form of a research studentship for this work.

References


