Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems

Larry Bull
Faculty of Computing, Engineering
& Mathematical Sciences
University of the Westof England
Bristol BS16 1QY

Toby O'Hara
Faculty of Computing, Engineering
& Mathematical Sciences
University of the West of England
Bristol BS16 1QY

Abstract

Learning Classifier Systems traditionally use a binary representation with wildcards added to allow for generalizations over the problem encoding. However, the simple scheme can be limiting in complex domains. In this paper we present results from the use of neural network-based representation schemes within the accuracy-based XCS. Here each rule's condition and action are represented by a small neural network, evolved through the actions of the genetic algorithm. After describing the changes required to the standard production system functionality, optimal performance is presented using multi-layered perceptrons to represent the individual rules. Results from the use of fuzzy logic through radial basis fuction networks are then presented. In particular, the new representation scheme is shown to produce systems where outputs are a function of the inputs.

1 INTRODUCTION

Since their inception Learning Classifier Systems (LCS) (Holland 1986) have been compared to neural networks, both conceptually (e.g. Farmer 1989) and functionally (e.g. Davis 1989, Dorigo & Bersini 1994, Smith & Cribbs 1994). In this paper we present a way to incorporate the neural paradigm into the accuracy-based XCS (Wilson 1995). LCS traditionally incorporate a binary rule representation, augmented with 'wildcard' symbols to allow for generalizations. This can become limiting in more complex domains (e.g. see (Schuurmans & Schaeffer 1989) for early discussions). Recently, a number of investigations have made use of other rule representations, including real numbers (Wilson 2000), messy GAs (Lanzi 1999a), logical S-expressions (Lanzi 1999b), and those where the output is a function of the input, including numerical S-expressions (Ahluwalia & Bull 1999) and fuzzy logic (e.g. Valenzuela-Rendon 1991).

We present a neural network-based scheme where each rule's condition and action are represented by a neural network. The weights of each neural rule being concatenated together and evolved under the actions of the genetic algorithm (GA)(Holland 1975). The approach is closely related to the use of evolutionary computing techniques in general to produce neural networks (see (Yao 1999) for an overview). In contrast to most of that work, an LCS-based approach is coevolutionary, the aim being to develop a number of (small) cooperative neural networks to solve the given task, as opposed to the evolution of one (large) network. That is, a decompositional approach to the evolution of neural networks is proposed. Moriarty and Miikulainen's SANE (1997) is most similar to the work described here, however SANE coevolves individual neurons to form a large network rather than small networks of neurons as rules.

2 X-NCS: A NEURAL LCS

2.1 XCS

In XCS rule-fitness for the GA is not based on rule predictions but on the accuracy of the predictions. The intention being to form efficient generalizations and a complete and accurate mapping of the search space (rather than simply focusing on the higher payoff niches in the environment).

On each time step match sets [M] are created. A system prediction is then formed for each action proposed by the rules in [M] according to a fitness-weighted average of the predictions of the rules. The system action is then selected, typically either deterministically (exploit) or randomly (explore). An action set [A] is then formed, the appropriate system output given and a reward may or not be received. If [M] is empty covering is used.

Reinforcement in XCS consists of updating three parameters, Error (*E*), Prediction (*p*), and fitness (*F*) for each appropriate rule. Each is updated every time it belongs to [A_{-1}] or [A] if it is a single step problem.

XCS uses a niche-GA (Booker 1985); the GA acts in action sets [A]. Two rules are selected based on fitness. In this paper we use a fixed size rule-base *N*; varying *N* as in (Wilson 1995) is not incorporated here. Rule replacement is based on the estimated size of each match set a rule participates in with the aim of balancing resources across niches. The GA is triggered (see also (Booker 1989)) within a given match set if the number of time steps since its last invocation in that set passes a fixed threshold, based on the average time-stamp of the rules. Typically this parameter is set to 25.

The reader is referred to (Butz & Wilson 2001) for full details of XCS. Wherever possible parameter values are in line with those used in (Wilson 1995) to facilitate comparisons with XCS using a ternary alphabet. In practice all parameter values lay within those used in (Wilson 1995) apart from two areas: population size and mutation rate which, as they relate to the higher number of real number genes in the genotype, are higher; and in function approximation, the relative value of the error parameter is a percentage value rather than being fixed. Further, we don't incorporate subsumption or maintain a rule for each action in a given [M].

All results in this paper are the average of ten runs.

2.2 NEURAL RULE REPRESENTATION

Each traditional condition-action rule is replaced by a single, fully connected neural network. All rules have the same number of nodes in their hidden layers (simplest case (Bull 2001)) and one more output node than there are possible actions. All weights are randomly initialized in the range { –1.0, 1.0 }, concatenated together in an arbitrary order and thereafter determined solely by the GA here.

The production system cycles through the same input-match-action-update cycle as the LCS, XCS in this case. However, since all rules explicitly 'see' all inputs, unlike the traditional scheme whereby defined loci can exclude certain rules from certain match-sets, the extra output node is added. This is used to signify membership of a given match-set. After the presentation of an input, each neural network rule produces a value on each of its output nodes in the appropriate manner, e.g. feedforward. If the extra 'not match-set member' node has the highest output value, the rule does not form part of the resulting match-set. In all other cases the rule forms part of the match-set, proposing the action corresponding to the output node with the highest activation. This matching procedure is repeated for all rules on each cycle.

Rule discovery operates in the same way as usual for XCS with real numbers (Wilson 2000). Hence the mutation operator is altered to adjust gene values using a normal distribution; small changes in weights are more likely than large changes upon satisfaction of the mutation probability (μ). The cover operator is altered such that when the match-set is empty, random neural networks are created until one gives its highest activation on an action node for the given input.

Results from using multi-layered perceptrons (MLPs) are now presented in well-known single-step and multi-step tasks. All nodes used a sigmoid transfer function.

3 A SINGLE-STEP TASK: 6-BIT MULTIPLEXER

We have tested the new rule representation on the two tasks used in (Wilson 1995), the first of which is a 6-bit version of the well-known, single-step multiplexer task. These boolean functions are defined for binary strings of length $l = k + 2^k$ under which the first k bits index into the 2^k remaining bits, returning the indexed bit.

In order to make analysis easier for each neural net an equivalent classifier was produced and recorded, though it must be emphasized that it played no part in the XCS processing, and was produced to measure the generality or specificity of the particular neural rule.

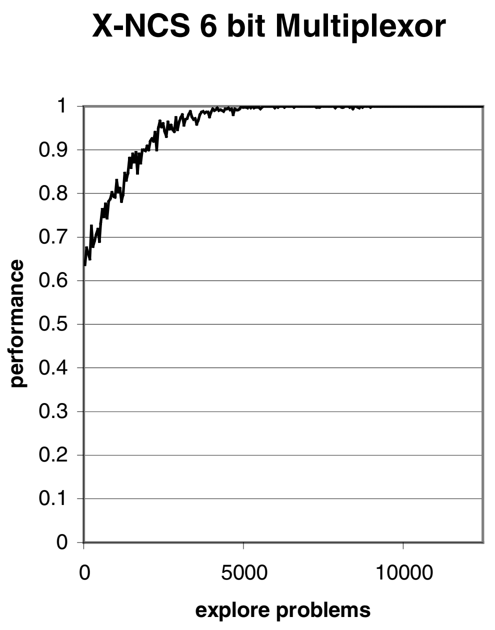

Figure 1: Performance of accuracy-based neural classifier system on the 6-bit multiplexer (l=6).

Figure 1 shows the results of using X-NCS on the single-step problem, averaged over ten runs, with all parameters as presented in (Wilson 1995) apart from the population size and mutation. That is, N=800, μ=0.08, β=0.2, ϕ=0.5, α=0.1, χ=0.8, θ=10, δ=0.1, $p1$=10.0, $F1$=10.0, ε1=0.0. As in (ibid.), payoff is given in 100 increments from 300/0 for each classification. Rules contain five nodes in their hidden layer.

From Figure 1 it can be seen that using the neural representation requires around 5000 exploit problems to solve the task, roughly equivalent to the binary representation (Wilson 1995). However, that an overhead may be incurred for a more complex representation may perhaps expected for such simple tasks. Analysis of the resulting rule-bases shows that, as well as the usual rules which match multiple inputs and propose a single action at a given payoff prediction level, multiple action rules emerge. That is, *for a given prediction level, accurate rules are evolved which suggest different actions depending on the input.*

4 A MULTI-STEP TASK: WOODS 2

Wilson (1995) presented the multi-step, and hence delayed reward, maze task Woods 2 to test XCS. Woods 2 is a toroidal grid environment containing two types of food (encoded 110 and 111), two types of rock (encoded 010 and 011) in regularly spaced 3 by 3 cells, and free space (000) (see (ibid.) for full details).

The learner is positioned randomly in one of the blank cells and can move into any one of the surrounding eight cells on each discrete time step, unless occupied by a rock. If it moves into a food cell the system receives a reward from the environment (1000) and the task is reset, i.e. food is replaced and the learner randomly relocated. On each time-step the learning system receives a sensory message, which describes the eight surrounding cells, ordered with the cell directly north and proceeding clockwise around it.

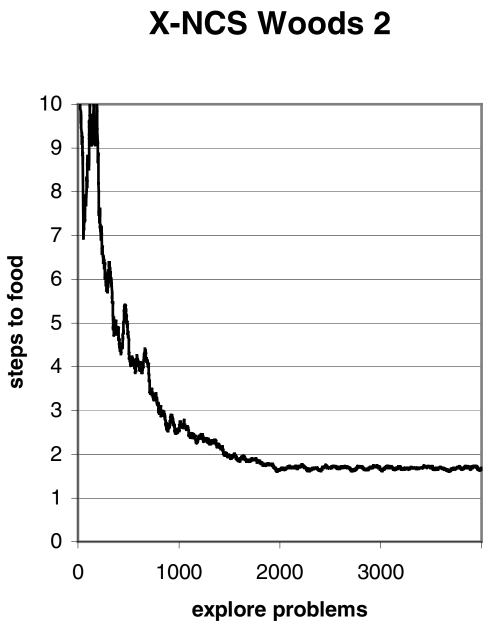

Figure 2: Performance of the accuracy-based neural classifier system in Woods 2.

Here, as in (Wilson 1995), the trial is repeated 8000 times, half explore and half exploit, and a record is kept of the moving average (over the previous 50 exploit trials) of how many steps it takes the system to move into a food cell on each trial.

Figure 2 shows results from using X-NCS in Woods 2 with all parameters as presented in (Wilson 1995) except, more rules and more mutation are used. That is, N=1600, μ=0.08, β=0.2, γ=0.71, ϕ=0.5, α=0.1, χ=0.8, θ=25, δ=0.1, p_I=10.0, F_I=10.0, ε_I=0.0. Rules again contained five hidden nodes. It can be seen that it takes the system around 2,000 explore problems, again roughly equivalent to the traditional encoding, to reach optimal performance (1.7 steps to food).

Analysis of the resulting rule-bases shows that neural rules emerge which have no error and produce different actions depending upon the input. Unlike in the multiplexer problem, we find that for some payoff levels these multi-action rules are more numerous than the equivalent single action rules. We presume that, if left to run for longer, the system would converge on a single neural rule for each payoff level; *maximal generalizations would be produced in both the condition and action space.*

As noted above, XCS usually forms generalizations for each action at each level of payoff. Within traditional reinforcement learning (Sutton & Barto 1998) a neural network is often used to produce generalizations for each possible action, where the networks are trained using gradient descent techniques. Under the scheme proposed here, X-NCS forms generalizations at a level between these two extremes using the GA to produce the neural networks. An advantage of this scheme over the other two is its ability to work with continuous action spaces. An application which exploits this last aspect of the representation scheme in a single-stepped task is now presented.

5 FUNCTION APPROXIMATION

It is well-known that multi-layered perceptrons with an appropriate single hidden layer and a non-linear activation function are universal classifiers (e.g. Hornick et al. 1989). Until recently LCS had not been used to solve tasks of the form $y = f(x)$ since their traditional representation scheme does not lend itself to such classes of problem. Fuzzy Logic LCS (see (Bonarini 2000) for an overview) represent, in principle, a production system-like scheme which can be used for such tasks but this remains unexplored. Ahluwalia and Bull (1999) presented a simple form of LCS which used numerical S-expressions for feature extraction in classification tasks. Here each rule's condition was a binary string indicating whether or not a rule matched for a given feature and the actions were S-expressions which performed a function on the input feature value. Most recently, Wilson (2001) has presented a form of XCS, termed XCSF, which uses piecewise-linear approximation for such tasks; using only explore trials all matching rules update their parameters, where such trials are run consecutively as a training period.

We have tested the neural rule representation for tasks of the form *y* = *f(x)*, where both *x* and *y* are real numbers between 0.0 and 1.0. The implementation estimated two functions, x-squared and a six variable root-mean-square. However, unlike the above mentioned work, the system requires very few changes to the design of the standard XCS system.

5.1 MODIFICATIONS TO X-NCS FOR FUNCTION APPROXIMATION

5.1.1 Processing of Real Numbers

The real number inputs were scaled between 0.4 and 0.8 to accommodate the lack of discrimination of the upper and lower end of the sigmoid function, as is usual in the use of MLPs. Output layer nodes are now linear.

5.1.2 Changes to Error Threshold Processing and System Error

In standard XCS, the error threshold ε_0 is a fixed fraction of the payment range. However with function approximation across a continuous (action) range, a fixed value may result in very inaccurate classifiers at the bottom end of the input range. It was therefore decided that ε_0 should be variable to enable the accuracy, and hence fitness, of the classifiers across the range to be equivalent. The variable value was chosen as the percentage of the target value at any particular point. The percentage chosen was 1% so, for example under x-squared, if the input was 0.3 the target output value (*f(x)*) would be 0.09. Here the required accuracy ε_0 would be 0.0009 and so classifiers that predicted within the range 0.0891 to 0.0909 would be given an accuracy of 1.0. In the same way, when the performance of the system is measured, the system error was calculated by taking the absolute difference between the target value and the prediction of the selected classifier, and dividing this by the target value, i.e. the system error is the percentage error between the target and the prediction value.

5.1.3 Match Set and Action Set Processing

The rule prediction value is taken from one output node of the individual's neural network. The selection of the match set is similar to before (Section 2), the only difference being that the 'not match-set member' output node merely has to have a positive value, rather than a value less than the primary output node, as above. The aim being to reduce the complexity of the task faced by individual rules.

In exploration all members of the match set are updated and rule discovery invoked if appropriate as per standard XCS. In XCS under exploitation, all classifiers which advocate the same action are put into the same set [A]. The chosen action set is the one which has the highest fitness weighted prediction. For function approximation we are looking for the rule whose prediction is most accurate, i.e. has the least error, and hence taking the classifier with the highest fitness weighted prediction would be inappropriate. Instead, the counterpart of prediction for such tasks is chosen, i.e. rule error, and so we choose the rule with the lowest value of error divided by fitness.

It was also found that for these function approximation tasks a biased uniform crossover operator (75%) appeared to give slightly better results than the single point crossover operator used above. This aspect of the system remains open to future investigation.

5.1.4 Rule Updating

The prediction value for each rule is taken as the value of the output of the neural network, i.e. the prediction value of the classifier can change at each iteration. By contrast, the error value of a rule is determined as per standard XCS. Accuracy is determined in the standard XCS way except, as mentioned above, the accuracy criterion is taken as a percentage of the current target value. Fitness is again calculated in the standard XCS way.

Thus the output value for a particular rule will change for each different input value. For example, for problem *n* with input value 0.3 -> prediction 0.0891, but problem *n*+1 with input 0.4 -> prediction 0.160. However the error value for each accurate classifier, although it varies as the predictions can deviate from their respective targets, is a small value that oscillates according to ε_0.

5.2 RESULTS FOR $Y=X^2$

In this task training consists of (alternating) 50,000 explore trials and 50,000 exploit trials each presenting a random input in the range [0.0, 1.0] scaled as mentioned above.

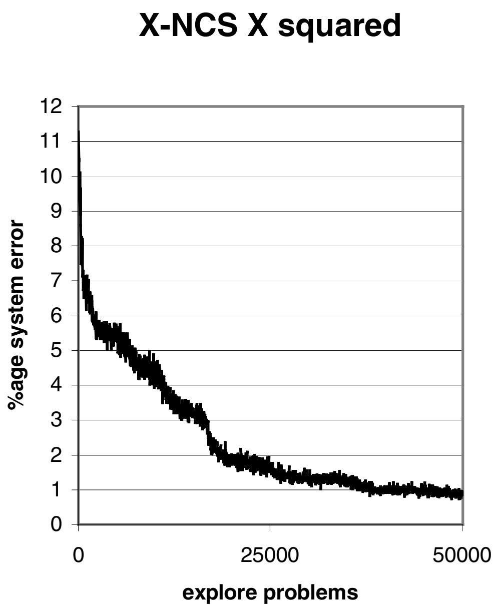

Figure 3: X-NCS on the x-squared function.

Figure 3 shows the performance of the accuracy-based neural classifier system on the x-squared function, averaged over ten runs, with a runnning average over the previous fifty exploit trials. The parameters used were: N=1000, β=0.2, φ=0.5, μ=0.03, α=0.1, χ=0.8, θ=10, δ=0.1, p_I=10.0, F_I=10.0, ε_I=0.01. Rules contained five hidden layer nodes. For the last 500 problems X-NCS is run in test mode and hence under the exploit scheme; after training with randomly generated examples the performance of the resulting system was tested using a number of unseen randomly generated examples.

From Figure 3 it can be seen that using the neural representation requires around 40,000 explore problems to solve the task, i.e. for the accuracy of the approximations to fall within 1% of the real *f(x)*.

Analysis of the resulting systems shows that better performance is achieved when one neural network emerges to cover the whole problem space, rather than through the co-operative sets seen above. The reasons for this appear two-fold: MLPs attempt to form global models by approximating between known data points; and the niche-based scheme of XCS encourages maximally general rules through increased chances to reproduce. This aspect of X-NCS will be returned to.

5.3 RESULTS FOR ROOT-MEAN-SQUARE

We have also examined the performance of the system on functions which contain more than one variable. Wilson (2001) presented a general, multi-dimensional function of the form $y = [(x1^2 + \ldots + xn^2) / n]^{½}$. We have used this "root mean squared" function with n=6, where training was identical to that of the x-squared task above.

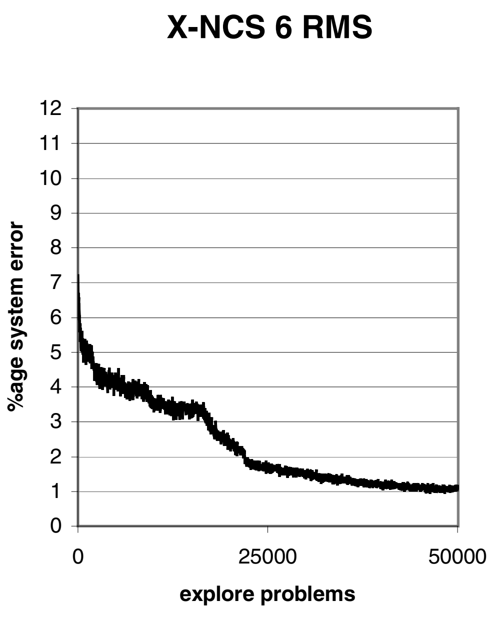

Figure 4: X-NCS on the 6 variable rms function.

The parameters used were: N=1000, β=0.2, φ=0.5, μ=0.09, α=0.1, χ=0.8, θ=10, δ=0.1, p_I=10.0, F_I=10.0, ε_I=0.01.

From Figure 4 it can be seen that using the neural representation requires around 50,000 explore problems to solve the task to an accuracy of 1%. As with the x-squared function, the most accurate solutions came from those in which one classifier covered the whole input range.

6 X-NFCS: A NEURO-FUZZY LCS

Radial basis function neural networks (RBFs) (e.g. Poggio & Girosi 1990), in contrast to MLPs, construct local function approximations using Gaussian functions processed in the hidden layer of the network. It was noted above (Sections 3 and 4) that in the discrete action tasks the MLP-based neural system formed traditional LCS coevolutionary solutions, whereas with continuous actions a single rule/network emerged. Hence the use of RBFs in X-NCS seems more likely to exploit the system's coevolutionary nature. There is another potential benefit to the use of RBFs.

The similarity between RBF networks and fuzzy rule-based systems is discussed in (Jang & Sun 1995). Fuzzy rule sets consist of membership functions over appropriate universes of discourse for input and output variables and rules which define input-output relations. The Gaussian functions of an RBF can be seen as fuzzy membership functions and the hidden layer nodes the fuzzy rules. In general, the benefits from combining neural computing with fuzzy logic are potentially large (see (Tsoukalas & Uhrig 1997) for an introduction). In this context it also avoids the possible need to alter the reinforcement process (see (Bonarini 2000) for discussions).

GAs have been used to evolve RBFs as they have MLPs. The most similar approach to that proposed here is Whitehead and Choate's (1995) scheme whereby the individual members of the population are the basis functions of a single network and heuristics tackle the competitor/cooperator problem.

Genomes are again strings of real numbers: the positions of the basis function centres, widths and weights of fully connected networks are concatenated in an arbitrary order to form the encoding. Output nodes (two) are again linear. Carse et al. (e.g. 2001) have proposed a crossover operator for fuzzy sets which alleviates the permutations (Radcliffe 1990) problem that can arise under the evolution of neural networks. That is, different genotypes can give the same phenotype and hence crossover may disrupt useful structures. The fuzzy logic crossover operator works in the input space rather than by postion on the genome. As with the MLPs, and perhaps due to the niche GA of XCS, the permutations aspect of the concatenated weights encoding does not appear to have been significant in the tasks explored here. All other system functionality is the same as in Section 5 – a positive response on the extra node means a rule doesn't join a given [M] and percentage error is used.

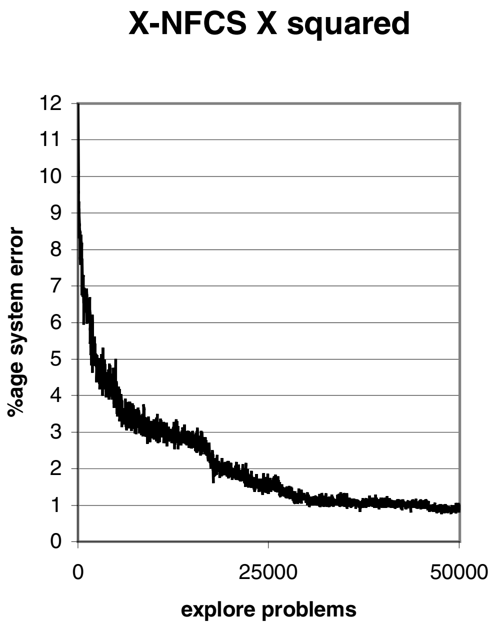

Figure 5: X-NFCS on the x-squared function.

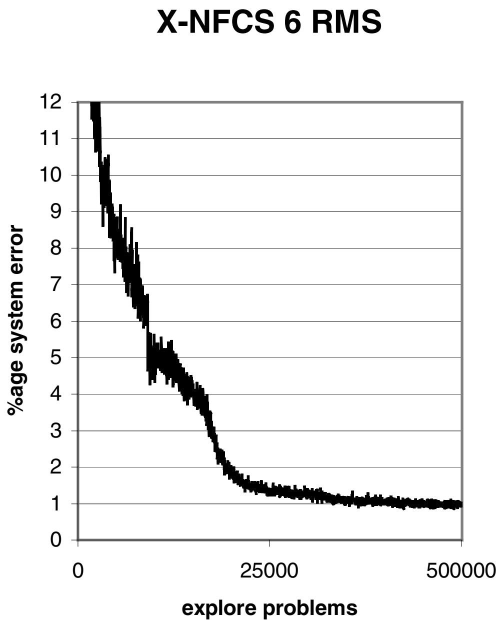

Figure 6: X-NFCS on the 6 variable rms function.

Figure 5 shows the performance of the accuracy-based neuro-fuzzy classifier system on the x-squared function, averaged over ten runs, with a runnning average over the previous fifty exploit trials. All parameters are as in section 5.2. It can be seen that the function is learnt to 1% accuracy around 30,000 explore problems. That is, a greatly reduced training period is required using the neuro-fuzzy system (compare with Figure 3). Analysis of the resulting systems shows that coevolutionary solutions appear, i.e. different rules emerge to handle different regions of the input space, together covering the total problem space.

Figure 6 shows the performance of X-NFCS on the six variable rms function. All parameters are as in section 5.3, but it was found necessary to use ellipsoid radial basis functions. Here, each input node maintains a radius for its Gaussian for each input node. It can be seen that accurate performance is obtained after 40,000 explore trials. Analysis again shows that a number of rules are used in the evolved systems and that the decompositional approach led to a reduced training period (compare with Figure 4).

Results (not shown) from the discrete action tasks of sections 3 and 4 also show optimal performance using RBFs, with learning times similar to the MLP-based system.

7 CONCLUSIONS

In this paper we have presented results from using a neural rule representation scheme within an accuracy-based learning classifier system. The effective combination of evolutionary computing and neural computing has long been an aim of machine learning (e.g. Belew et al 1989). It is our aim to exploit the coevolutionary and accuracy processes of XCS to realize such systems. Hopefully, this will also ease the use of LCS in more complex problem domains.

We are currently examining the use of the system for more complex tasks, with discrete or continuous action spaces, both single-step and multi-step. For the latter we are also exploring the use of recurrent connections with the neuro and neuro-fuzzy systems for non-Markov domains (after (Bull & O'Hara 2001)).

References

Ahluwalia, M. & Bull, L. (1999) A Genetic Programming Classifier System. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds*)* *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99.* Morgan Kaufmann, pp11-18.

Belew, R.K., McInerney, J. & Schraudolph, N.N. (1989) Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. In C.G. Langton, C. Taylor, J.D. Farmer & S. Rasmussen (eds) *Artificial Life II,* Addison-Wesley, pp511-548.

Bonarini, A. (2000) An Introduction to Learning Fuzzy Classifier Systems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp83-106.

Booker, L. (1985) Improving the Performance of Genetic Algorithms in Classifier Systems. In J.J. Grefenstette (ed.) *Proceedings of the First International Conference on Genetic Algorithms and their Applications,* Lawrence Erlbaum Assoc., pp80-92.

Booker, L. (1989) Triggered Rule Discovery in Classifier Systems. In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, pp265-275.

Bull, L. (2001) A Brief note on the use of Constructivism in NCS. *UWE Learning Classifier Systems Group Technical Report UWELCSG01-006*. Available from http://www.cems.uwe.ac.uk/lcsg.

Bull, L. & O'Hara, T. (2001) NCS: A Simple Neural Classifier System. *UWE Learning Classifier Systems Group Technical Report UWELCSG01-003*. Available from http://www.cems.uwe.ac.uk/lcsg.

Butz, M. & Wilson, S.W. (2001) An Algorithmic Description of XCS. *Advances in Learning Classifier Systems: Proceedings of the Third International Conference – IWLCS2000*. Springer, pp253-272.

Carse, B, Fogarty, T.C, Patel, & Sullivan, J (2001) Evolution and Learning in Radial Basis Function Neural Networks- A Hybrid Approach. In M.J V. Honavar & K. Balakrishnan (eds) *Advances in the Evolutionary Synthesis of Intelligent Agents.* MIT Press, pp247-272.

Davis, L. (1989) Mapptin Neural Networks into Classifier Systems. In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, pp375-378.

Farmer, D. (1989) A Rosetta Stone for Connectionism. *Physica D* 42:153-187.

Holland, J.H.(1975) *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Holland, J.H. (1986) Escaping Brittleness. In R.S. Michalski, J.G. Carnoell & T.M. Mitchell (eds) *Machine Learning: An Artificial Intelligence Approach 2*. Morgan Kaufmann, pp48-78.

Hornick, K., Stinchombe, M. & White, H. (1989) Multiayer Feedforward Networks are Universal Approximators. *Neural Networks 2*(5): 359-366.

Jang, J-S & Sun, C-T. (1995) Neuro-fuzzy Modeling and Control. *Proceedings of the IEEE* 83(3): 378-406.

Lanzi, P-L (1999a) Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds*) Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99*. Morgan Kaufmann, pp11-18.

Lanzi, P-L (1999b) Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds) *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99*. Morgan Kaufmann, pp11-18.

Moriarty, D.E. & Miikulainen, R. (1997) Forming Neural Networks Through Efficient and Adaptive Coevolution. *Evolutionary Computation* 5(2): 373-399.

Poggio, T. & Girosi, F. (1990) Networks for approximation and learning *Proceedings of the IEEE*, 78:1481-1497.

Radcliffe, N.J. (1990) Genetic Set Recombination and its Application to Neural Network Topology Optimisation. Technical Report EPCC-TR-91-21, University of Edinburgh, Scotland.

Schuurmans, D. & Schaeffer, J. (1989) Representational Difficulties with Classifier Systems. In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, pp328-333.

Smith, R.E. & Cribbs, H.B. (1994) Is a Learning Classifier System a Type of Neural Network*? Evolutionary Computation* 2(1): 19-36.

Sutton, R. & Barto A. (1998) *Reinforcement Learning*. MIT Press.

Tsoukalas, L.H & Uhrig, R.E. (1997) *Fuzzy and Neural Approaches in Engineering*. Wiley.

Valenzuela-Rendon, M. (1991) The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In L. Booker & R. Belew (eds*) Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp346-353.

Whitehead, B.A. & Choate, T.D. (1994) Evolving Space-filleing Curves to Distribute Radial Basis Functions over an Input Space. *IEEE Transactions on Neural Networks* 5(1): 15-23.

Wilson, S.W. (1995) Classifier fitness based on accuracy. *Evolutionary Computation* 3(2): 149-175.

Wilson, S.W. (2000) Get Real! XCS with Continuous-Valued Inputs. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp209-222.

Wilson, S.W. (2001) Function Approximation with a Classifier System. In L. Spector. E.D Goodman, A. Wu, W.B. Langdon, H-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon & E. Burke (eds*) Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-2001*. Morgan Kaufmann, pp974-984.

Yao, X. (1999) Evolving Artificial Neural Networks. *Proceedings of the IEEE* 87(9): 1423-1447.