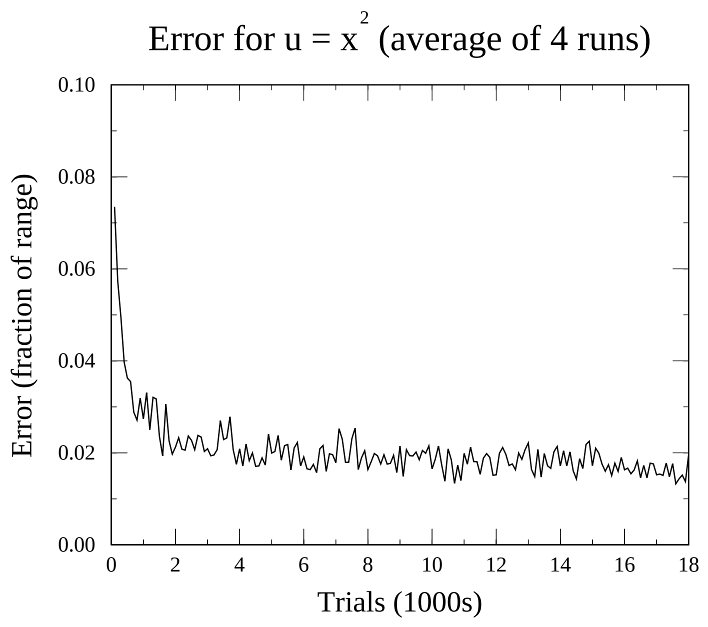

Error for u = x 2 (average of 4 runs)
Error (fraction of range)
0.10
0.08
0.06
0.04
0.02
0.00
0
2
4
6
8
10
12
14
16
18
Trials (1000s)

Figure 1

as yet little further analysis.

Our work on the mapping problem has reached a level well beyond the starting point, but has not much dented the basic questions: can a classifier system adapt its effective resolution to the mapping's curvature; will the resources required for high-dimensional mappings exhibit the increasing efficiency predicted by theory (not discussed here)? The second question is to some extent dependent on the first. We hope by the time of the Workshop to have some better answers.

References

Albus, J. S. (1975). A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control,* Trans. ASME, Series G, Vol. 97, No. 3, Sept. 1975.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley.

Hinton, G. E. (1981). Shape representation in parallel systems. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 1088-1096). Los Altos, CA: William Kaufmann, Inc.

Poggio, T. & Edelman, S. (1990). A network that learns to recognize three-dimensional objects. *Nature,* **343**, 263-266.

Valenzuela-Rendón, M. (1991). The fuzzy classifier system: a classifier system for continuously varying variables. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 346-353). San Mateo, CA: Morgan Kaufmann.

smallest level within a few thousand trials. Match sets tended to have 10-15 members. Most classifiers had receptive fields of three resolution units (field widths of 1, 3, 5,...,R are possible). The receptive fields of a match set were typically centered on two or three adjacent input resolution intervals. The system seems to drive toward “narrow” (field) classifiers that each contain a close approximation to the correct answer, rather than toward “broad” field classifiers that produce an accurate answer collectively (by averaging outputs) but are not individually very accurate. This quite pronounced tendency toward individual accuracy would seem due to the essentially competitive nature of the GA coupled with our payoff algorithms: individually accurate classifiers appear to have the best survival chances.

In step (2) we set out to determine whether classifier field sizes and degree of overlap (i.e., placement of centers) would vary with the curvature of the mapping. To maintain a constant error, the system should employ narrower and/or more highly overlapping classifiers where the curvature is steeper. Unfortunately, our experiments up to now have not demonstrated such a variation to a significant extent, despite several modifications of the payoff algorithm. For example, for the function $u = x^2$, and an evolved population that had learned under the payoff algorithm above, a scan of x from 0.0 to 1.0 shows a linearly increasing error in u. This is consistent with absence of field-size variation over the domain, since the slope of the function is $2x$.

It is of course important to get varying field sizes, since that is the basic point of evolving the classifiers. We would like the system to produce broad classifiers—and thus need fewer of them—where the mapping is only slightly curved, concentrating its resources in the steep parts which are harder to approximate. The matter is currently in thought.

We have done a few step (3) experiments. On the 2-D mapping $u = (x^2 + y^2)/2$ the system produces results similar to and consistent with those on $u = x^2$. The error goes rapidly to a level about twice that of the 1-D case; this is consistent with the fact that inputs x and y are independent. Error also appears to rise over the domain proportional to the partial derivatives, indicating that field sizes are again not adjusting to curvature. The system works for 2-D problems, but we have thus each classifier asserts directly what the system's output should be. Second, we found that a strength-weighted average of the weights produces significantly better results than an unweighted average. Third, it is better to adjust weights gradually, that is, to change a weight toward but not all the way to, the correct value each time the weight is adjusted. This causes the weights to settle more or less optimally with respect to the mapping and the receptive field size, indirectly reducing strength fluctuations to which the GA is sensitive.

Choice of a good strength-adjustment technique proved rather subtle. Each classifier in a match set makes its own assertion as to the correct output for that input. Since the correct output is known, one can immediately calculate each classifier's absolute error for that trial and use the inverse—call it the accuracy—as a payoff. We found that more stable performance results from dividing the individual payoffs by the number of classifiers in the match set: the sharing discourages population takeover drives by the currently best classifiers, a technique employed in other classifier systems. In this system, because a classifier's error can be arbitrarily small (if it's "lucky"), the payoff can be extremely large. We found it advantageous, also for stability, to place a ceiling on the inverse of the error approximately equal to the underlying resolution R.

The best performance, however, was not found until the strength-adjustment contained some form of "punishment". An effective algorithm, employed in Figure 1, gives payoff as above just to the *top* most accurate classifiers in the match set, where *top* is a constant (e.g., 5) and is also the divisor used for sharing. The rest of the matchset sustains relatively large (e.g., 50%) strength reductions.

The genetic algorithm was applied on each input trial with a probability of 0.5. If invoked, two classifiers were selected based on strength, crossed with probability 0.6 and mutated with probability .01 per bit, then inserted into the population with a very low initial strength. Two classifiers were deleted based on the inverse of their strength.

The results in step (1) are exemplified by Figure 1. Error reached roughly twice the optimally and each matching classifier also received a strength adjustment. A genetic algorithm step was invoked periodically in which two offspring were produced and two population members were deleted. The population size was 200.

The measure of performance was the absolute difference between the system's output and the correct value, expressed as a percentage of the output range, averaged over the last fifty trials. Figure 1 shows a performance curve for the mapping $u = x^2$. Note that the error rapidly falls to 0.02 (or 2%). What is the minimum possible error? In this experiment the input domain was quantized into R = 32 equal parts, so that with random inputs, each quantized interval represented the true input value with an average absolute error of 1/4R, or in this case 1/128 = .008, or about 1%. This assumes that an interval represents its center value; if inputs occur randomly over the interval, their average absolute difference from the center value will be one-fourth of the interval length. In the experiment of Figure 1, the minimum possible error is then the average interval error times the function slope (i.e. $2x$) integrated over the domain, or again, 0.008.

Our experiments have three stepwise objectives: (1) first get it to work, i.e., produce results with low errors on simple 1-D mappings; (2) on non-linear 1-D mappings, see what is required for the system to assign, e.g., small-field, closely spaced classifiers to regions of high curvature to maintain accuracy in those regions; (3) test on non-linear 2-D mappings for curvature following, and compare the number of classifiers required for a given average error to the size of a look-up table solution having the same average error. This last is a measure of coding—actually coarse-coding—efficiency which the work of Albus and others (Hinton 1981) suggests can rise dramatically as dimensionality increases.

In step (1) we learned a number of things. In the first place, we found that it is *not* desirable simply to add up the weights to produce the output (as occurs in, e.g., the perceptron). In contrast to systems that are not undergoing structural evolution, in our system the number of classifiers in the matchset for a given input is in general constantly changing, which introduces spurious noise if the weights are simply added up. The problem was solved by taking the average of the weights;

an output value. For example, let one device match (become active) if ($0.0 < x < 0.1$) and ($0.8 < y < 0.9$), and let its output assertion be "0.9"; this "classifier" (as we shall call them) could be part of a system that performed the mapping $\{x, y \rightarrow u \ni u = x + y\}$. Over its sub-region, the classifier could have been trained to assert an approximation to the mapping's correct output. A population of such devices could handle the whole mapping. But—how should the receptive field sizes and positions of the population's many "classifiers" be set in order to satisfy criteria of accuracy and efficiency? The hypothesis of this work is that this can be done by treating the population as a kind of classifier system which determines the field sizes and positions adaptively.

In our approach we define a classifier whose condition part contains a binary representation of the position of a receptive field center and its size (or "spread"), for each of the domain variables in the mapping to be approximated. Real-valued centers and spreads could have been used, but instead we quantized the input domain and used binary encoding because binary GA methods are better known. The classifier's output is a scalar weight that is adjustable by standard error correction methods, such as the Widrow-Hoff algorithm. (The weights could have been evolved, too, but we wished to concentrate on evolution of receptive field structure.) When presented with an input vector, the output of the system as a whole is just some linear combination of the output weights of all classifiers that match that input.

To standardize, we used a unit input domain. The output range could have any extent. However, error on a given output dimension was measured as a percentage of that dimension's output range. A mapping $\{\Re^m \rightarrow \Re^n\}$ was conceived as a set of m separate mappings from $\Re^n \rightarrow \Re$. The reason is that the functions involved in the separate component mappings are in general different, with different curvature characteristics, so that optimal receptive field sizes and positions will also be different.

In a cycle of the basic experiment, random inputs were provided, a match set determined, and the weights of the matching classifiers combined linearly to produce the system output. The weights were then adjusted depending on what the correct output was ("supervised" learning),

Classifier System Mapping of Real Vectors

Stewart W. Wilson
The Rowland Institute for Science
100 Cambridge Parkway
Cambridge, MA 02142
(wilson@smith.rowland.org)

Submitted to the International Workshop on Learning Classifier Systems

Extended Abstract

Mappings from real vectors to real vectors $\{\Re^m \rightarrow \Re^n\}$ are important for sensory-motor control in natural and artificial systems, and even play a role in certain models of higher cognitive function. Such mappings have been studied using neural networks (Hertz, Krogh & Palmer 1991), and the topic is important in the field known as approximation theory (Poggio 1990). As early as 1975, Albus introduced a learning technique for mappings known as CMAC (Albus 1975) that was inspired by the wiring of the cerebellum and is applicable to robotic control. Valenzuela's (1991) fuzzy classifier system learns real vector mappings.

While the training of real vector mappings is now quite well understood, most systems start with a fixed structure or fixed structural parameters, then modify weights. A largely missing ingredient is automatic methods of obtaining efficiency in the sense of minimizing the structure or number of working elements subject to an error criterion. Since nearly every mapping, however tortuous, has a piecewise-linear approximation, one can imagine a system that automatically adapts its structure to the mapping's curvature, using fewer and coarser elements where the curvature is small but more and narrower elements where it isn't. Our approach to this objective is to try to combine the approximation philosophy of CMAC with the adaptive possibilities of a classifier system.

Consider a device with "receptive fields" that can match a sub-region of a domain and assert