Function Approximation with a Classifier System

Stewart W. Wilson
Prediction Dynamics, Concord MA 01742
Department of General Engineering
The University of Illinois at Urbana-Champaign IL 61801
wilson@prediction-dynamics.com

Abstract

A classifier system, XCSF, is introduced in which the prediction estimation mechanism is used to learn approximations to functions. The addition of weight vectors to the classifiers allows piecewise-linear approximation. Results on functions of up to six dimensions show high accuracy. An interesting generalization of classifier structure is suggested.

1 Introduction

Classifier systems estimate payoff. That is, the rules (classifiers) evolved by a classifier system each keep a statistical estimate of the payoff (reward, reinforcement) expected if the classifier's condition is satisfied and its action is executed by the system. In XCS, a particular classifier system architecture (Wilson 1995), the classifiers form quite accurate payoff estimates, or *predictions*, since the classifiers' fitnesses under the genetic algorithm depend on their prediction accuracies. In effect, XCS approximates the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, where X is the set of possible inputs, A the system's set of available actions (typically finite and discrete), and P is the set of possible payoffs. If attention is restricted to a single action $a_i \in \mathrm{A}$, the mapping has the form $\mathrm{X} \times a_i \Rightarrow \mathrm{P}$, which is a function from input vectors x to scalar payoffs. Thus the system approximates a separate function for each a_i.

In the reinforcement learning contexts in which classifier systems are typically used, the reason for forming these payoff function approximations is to permit the system to choose, for each x, the best (highest-paying) action from A. However, there are contexts where the output desired from a learning system is not a discrete action but a continuous quantity. For instance in predicting continuous time series, the output might be a future series value. In a control context, the output might be a vector of continuous quantities such as angles or thrusts. Apart from classifier systems based on fuzzy logic (Valenzuela-Rendón 1991; Bonarini 2000), there are none which produce real-valued outputs. Our hypothesis was that the payoff function approximation ability of XCS could be adapted to produce real-valued outputs, as well as be used for function approximation in general applications. Our objective in this paper is to examine, as a first step, XCS's potential for learning approximations to simple functions.

In the paper we adapt XCS to learn functions of the form $y = f(x)$, where y is real and x is a vector with integer components $x_1, ... x_n$. Our results demonstrate approximation to high accuracy, together with evolution of classifiers that tend to distribute themselves efficiently over the input domain. As a by-product of the research, a conceptual generalization of classifier structure is developed.

The next section describes modifications of XCS for function approximation. Section 3 has results on a simple piecewise-constant approximation. In Section 4 we introduce a new classifier structure that permits piecewise-linear approximations. Results on simple functions are shown in Section 5. In Section 6 we demonstrate accurate approximation of a six-dimensional function. Section 7 presents the classifier structure generalization. The final section has our conclusions and suggestions for future work.

2 Modification of XCS

XCS was modified in two respects (later, a third). The first was to adapt the program for integer instead of binary input vectors. The second, very simple, was to make the program's payoff predictions directly accessible at the output and to restrict the system to a single (dummy) action. The resulting program was called

XCSF. (We omit a description of basic XCS, but refer the reader to Wilson (1995), Wilson (1998), and the updated formal description in Butz and Wilson (2001) that XCSF follows most closely.)

The changes to XCS for integer inputs were as follows (drawn from Wilson (2001)). The classifier condition was changed from a string from {0,1,#} to a concatenation of "interval predicates", $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are integers. A classifier matches an input x with attributes x_i if and only if $l_i \leq x_i \leq u_i$ for all x_i.

Crossover (two-point) in XCSF operates in direct analogy to crossover in XCS. A crossover point can occur between any two alleles, i.e., within an interval predicate or between predicates, and also at the ends of the condition (the action is not involved in crossover). Mutation, however, is different. The best method appears to be to mutate an allele by adding an amount $\pm rand(m_0)$, where m_0 is a fixed integer, $rand$ picks an integer uniform randomly from $(0, m_0]$, and the sign is chosen uniform randomly. If a new value of l_i is less than the minimum possible input value, in the present case 0, the new value is set to 0. If the new value is greater than u_i, it is set equal to u_i. A corresponding rule holds for mutations of u_i.

The condition of a "covering" classifier (a classifier formed when no existing classifier matches an input) has components $l_0, u_0, ..., l_n, u_n$, where each $l_i = x_i - rand_1(r_0)$, but limited by the minimum possible input value, and each $u_i = x_i + rand_1(r_0)$, limited by the maximum possible input value; $rand_1$ picks a random integer from $[0, r_0]$, with r_0 a fixed integer.

For the subsumption deletion operations, we defined subsumption of one classifier by another to occur if every interval predicate in the first classifier's condition subsumes the corresponding predicate in the second classifier's condition. An interval predicate subsumes another one if its l_i is less than or equal to that of the other and its u_i is greater than or equal to that of the other. For purposes of action-set subsumption, a classifier is more general than another if its *generality* is greater. Generality is defined as the sum of the widths $u_i - l_i + 1$ of the interval predicates, all divided by the maximum possible value of this sum.

3 Piecewise-Constant Approximation

The simplest way to approximate the function $y = f(x)$ with XCSF is to let x be the input and y the payoff. After sufficient sampling of the input space, the system should, given an x, more or less accurately predict the corresponding y. In all XCS-like systems the fitness of a classifier depends on its accuracy of prediction, so that XCSF should converge to a population of classifiers that, over their respective input ranges, predict payoff well. The closeness of the approximation should be controllable with the *error threshold* ϵ_0, as follows. A classifier with *prediction error* ϵ_1 has higher *accuracy* than a classifier with error ϵ_2 if $\epsilon_1 < \epsilon_2$ (Butz and Wilson 2001). Since classifier *fitness* depends on accuracy, classifiers with lower errors will win out in the evolutionary competition. However, by definition, classifiers with errors *less* than ϵ_0 have constant fitness, so no further fitness pressure applies. Thus ϵ_0 should limit the closeness of the approximation.

It is important that besides evolving accurate classifiers, the system employ the classifiers efficiently over the input domain. In slowly-changing or low-gradient regions of the function we would hope to evolve classifiers with relatively large interval predicates. The function value, and thus a given classifier's error, would change relatively little over such regions; so, as in XCS, a tendency toward accurate, maximally general conditions (Wilson 1995) should cause the interval predicates to expand. Conversely, we should expect classifiers with small interval predicates where the function is changing rapidly. We also hope for an efficient *distribution* of classifiers over the domain in the sense that the tiling minimizes overlaps between classifier predicates.

Figure 1 shows typical results from an experiment in which XCSF approximated the function $y = x^2$, a parabola, over the interval $0 \leq x < 100$. (In all experiments reported here, the input value range was [0,99].) Plotted are XCSF's prediction for each possible x as well as the function itself.

The system learned from a dataset consisting of 1000 x, y pairs, with x chosen randomly and y the corresponding function value. In the experiment, a pair was drawn randomly from the dataset, its x value presented to XCSF as input, and the y value used as reward or reinforcement. XCSF formed a match set [M] of classifiers matching x and calculated the *system prediction* for each possible action in the usual way; since there was only one, dummy, action, just one system prediction was calculated and that became the system's output. An *action set* [A] was formed consisting of classifiers in [M] having the dummy action (i.e., all classifiers in [M]), and the predictions of the classifiers in [A] were adjusted using the reward, y, in the usual way; the other classifier parameters were also adjusted. A genetic algorithm was run in [A] if called for. This cycle was repeated 50,000 times after which the plot in Figure 1 was obtained by sweeping through all pos-

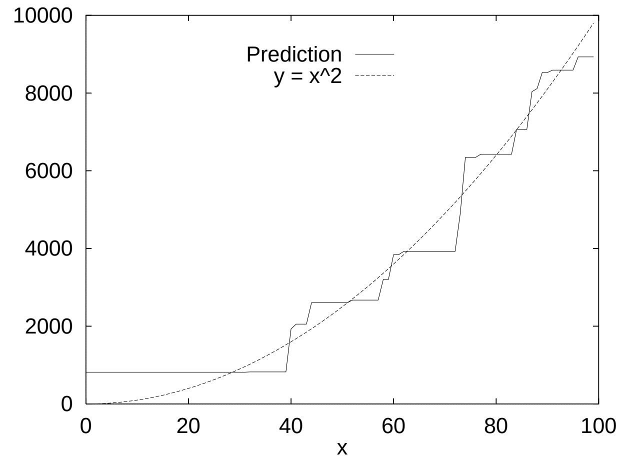

Figure 1: Piecewise-constant approximation to the parabola function $y = x^2$. $\epsilon_0 = 500$.

```
          CONDITION          PRED   ERR   FITN NUM

  0. |..................o0| 8992. 283. .796 31
  1. |.................0o.| 7876. 322. .906 32
  2. |..............o00...| 6162. 503. .564 20
  3. |..............000...| 5990. 570. .188  9
  4. |.............o000...| 5988. 569. .028  1
  5. |.............o00o...| 5668. 413. .075  3
  6. |.............o00....| 5534. 281. .090  3
  7. |.............o0o....| 5423. 205. .298 10
  8. |............o000o...| 5053. 524. .002  1
  9. |............o00o....| 4861. 465. .054  5
 10. |..........000o......| 3854. 468. .862 30
 11. |......o0000.........| 1481. 364. .135  7
 12. |......00000.........| 1359. 377. .358 18
 13. |000000000o..........|  821. 378. .372 15
 14. |00000000o...........|  820. 377. .371 15
```

Figure 2: Classifiers from experiment of Figure 1. (PREDiction, ERRor, FITNess, NUMerosity.)

sible values of x and recording the resulting system predictions.

Parameter settings for the experiment were as follows, using the notation of Butz and Wilson (2001): population size $N = 200$, learning rate $\beta = 0.2$, error threshold $\epsilon_0 = 500$, fitness power $\nu = 5$, GA threshold $\theta_{GA} = 48$, crossover probability $\chi = 0.8$, mutation probability $\mu = 0.04$, deletion threshold $\theta_{del} = 50$, fitness fraction for accelerated deletion $\delta = 0.1$. In addition: mutation increment $m_0 = 20$ and covering interval $r_0 = 10$. GA subsumption was enabled, with time threshold $\theta_{GAsub} = 50$.

Several aspects of Figure 1 are of interest. The Prediction curve has a "staircase" appearance typical of a piecewise-constant approximation. The height of the major "steps" varies between about 1000 and 2000. Examination of individual steps indicates an average error roughly consistent with the value of ϵ_0, suggesting that ϵ_0 is controlling the closeness of the approximation. The width of the steps is, again roughly, wider in the "flatter" part of the function and narrower in the steep part. Finally, it is significant that the Prediction curve indeed takes the form of a staircase, instead of being smoother. Long flat "steps" suggest that one set of classifiers is in control (forms [M]) after which, on the next step, another set takes over. This in turn suggests a tendency toward efficient distribution of classifier resources over the domain.

Figure 2 gives another perspective. It is a listing of the (macro)classifiers of the population at the end of the experiment, 15 in all. Shown are each classifier's condition, prediction, error, fitness, and numerosity. Note that most classifiers with substantial fitnesses have errors less than ϵ_0. A special graphic notation is used to represent the condition. Since x has just one component, the condition contains just one interval predicate. The possible range of x, 0-99, is divided into 20 equal subranges. An interval predicate is indicated by a cluster of "`0`"s that covers its range. If an interval predicate entirely covers a subrange, e.g., 35-39, an "`0`" is placed at that range's position. If the interval predicate covers some of but less than the whole of a subrange, a small "`o`" is put there. This notation has been found more perspicuous than using the raw numbers.

Note how, consistent with Figure 1, the classifier conditions are larger toward the beginning of the domain, where the function slope is lower. It is also interesting that the classifiers with higher fitnesses and numerosities cover the domain without a great deal of overlap. These classifiers dominate the calculation of the system prediction, since the latter is a fitness-weighted average of the predictions of matching classifiers. Because they dominate, the Prediction curve takes the form of a staircase. Apart from the presence of the remaining, lower fitness, classifiers, the distribution of resources over the domain is thus relatively efficient.

In sum, XCSF succeeds in approximating the function in accordance with a stated error criterion (confirmed for additional values of ϵ_0) and the classifiers are employed reasonably well. Still, a piecewise-constant approximation is primitive compared with an approximation where the approximating segments more closely follow the function's contour. The simplest such approach is a *piecewise-linear* approximation. But how could a piecewise-linear approximation be done with a classifier system?

4 Piecewise-linear Approximation

Traditionally, a classifier's prediction is a number intended to apply for all inputs x that satisfy its condition. However, for function approximation, it would be desirable if the prediction could vary over the condition's domain, since the function being approximated generally varies. In effect, the prediction itself should be a function, the simplest form of which would be a linear polynomial in the input components, call it $h(x)$. The function $h(x)$ would substitute for the classifier's traditional (scalar) prediction, p. Then, given an input x, each matching classifier would *calculate* its prediction by computing $h(x)$.

For approximating a one-dimensional function $f(x)$, $h(x)$ would be a two-term polynomial $h(x) = w_0 + w_1x_1$. In this case, w_1 can be thought of as the slope of an approximating straight line, with w_0 its intercept. For an n-dimensional $f(x)$, $h(x) = w \cdot x'$, where w is a *weight vector* $(w_0, w_1, ..., w_n)$ and x' is the input vector x augmented by a constant x_0, i.e., $x' = (x_0, x_1, ..., x_n)$. In this case $h(x)$ computes a *hyperplane approximation* to $f(x)$. Classifiers would have different weight vectors w since in general the domains of their conditions differ.

Of course, the classifiers' weight vectors must be adapted. If classifiers are to predict with a given accuracy, the coefficients w_i of their weight vectors must be appropriate. One approach is use an evolutionary algorithm. The weight vector would be evolved along with the classifier condition. For this, the w_i could be concatenated with the interval predicates of the condition and the whole thing evolved as a unit. Or, it might be preferable to use separate processes: there is reason to think the *Evolutionsstrategie* might be more suitable than the GA for the weight vector. In the present work, however, we did not use an evolutionary technique for the weight vector, but instead adapted it using a modification of the *delta rule* (Mitchell 1997).

The delta rule is given by

$$\Delta w_i = \eta(t - o)x_i,$$

where w_i and x_i are the ith components of w and x', respectively. In the quantity $(t - o)$, o is the output, in the present case the classifier prediction, and t is the *target*, in this case the correct value of y according to $y = f(x)$. Thus $(t - o)$ is the amount by which the prediction should be corrected (the negative of the classifier's instantaneous error). Finally, η is the *correction rate*. The delta rule says to change the weight proportionally to the product of the input value and the correction.

Notice that correcting the w_i in effect changes the output by

$$\Delta o = \Delta w \cdot x' = \eta(t - o)|x'|^2.$$

Because $|x'|^2$ is factored in, it is difficult to choose η so as to get a well-controlled overall rate of correction: η too large results in the weights fluctuating and not converging; if η is too small the convergence is unnecessarily slow. After some experimentation with this issue, we noticed that in its original use (Widrow and Hoff 1988), the correction rate was selected so that the entire error was corrected in one step; this was possible, however, because the input vector was binary, so its absolute value was a constant. In our problem, reliable one-step correction would be possible if a *modified delta rule* were employed:

$$\Delta w_i = (\eta/|x'|^2)(t - o)x_i.$$

Now the total correction would be strictly proportional to $(t - o)$ and could be reliably controlled by η. For instance, $\eta = 1.0$ would give the one-step correction of Widrow and Hoff. In the experiments that follow, we used the modified delta rule with various values of $\eta \leq 1.0$.

Use of a delta rule requires selection of an appropriate value for x_0, the constant that augments the input vector. In tests, we found that if x_0 was too small, weight vectors would not learn the right slope, and would tend to point toward the origin. Choosing $x_0 = 100$ solved this problem, perhaps because it was then of the same order of magnitude as the other x_i.

For piecewise-linear approximation, no changes were necessary to XCSF except for addition of the weight vectors to the classifiers, and provision for calculation of the predictions and application of the modified delta rule to the action set classifiers on every time-step. In a classifier created by covering, the weight vector was randomly initialized with weights from [-1.0,1.0]; GA offspring classifiers inherited the parents' weight vectors. Both policies yielded performance improvements over other initializations. In the experiments, most parameter settings were the same as those given in Section 3; differences will be noted. Settings of the new parameter, η, will be given.

5 Tests on Simple Functions

Preliminary testing was carried out approximating functions that were themselves linear or piecewise linear. For example, tests were done on the function "2-line", defined as

$$\begin{aligned} y &= 50x + 1000, \quad 0 \leq x < 50 \\ &= 130x - 3000, \quad 50 \leq x < 100. \end{aligned}$$

Parameters for the experiment were the same as pre-

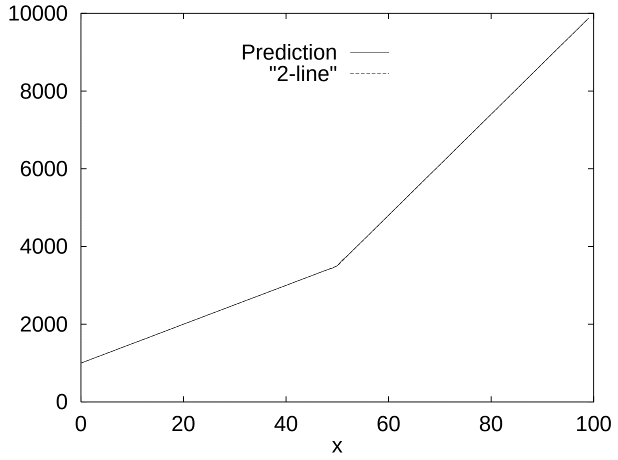

Figure 3: Piecewise-linear approximation to piecewise-linear function "2-line". $\epsilon_0 = 10$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|.........o0000000000|	4705.	19.	.158	62
1.	|..........o000000000|	4670.	0.	.706	274
2.	|..........o000000000|	4670.	0.	.044	18
3.	|..........0000000000|	4670.	0.	.090	29
4.	|00000000000o.........|	3057.	16.	.005	6
5.	|00000000000o.........|	3050.	0.	.676	247
6.	|000000000o..........|	3050.	0.	.377	164

Figure 4: Classifiers from experiment of Figure 3

viously, except for $N = 800$, $\epsilon_0 = 10$, $\theta_{GAsub} = 100$, and $\eta = 0.4$. The approximation obtained (Figure 3) was so close that the plots of the prediction and the function itself are difficult to distinguish visually.

Figure 4 shows the seven classifiers at the end of the run[1]. They are clearly divided into one group for the upper segment of the function and another for the lower segment. The dominant classifier in the upper group, no. 1, has error zero and a predicate covering the interval 52-99 (inclusive). In the lower group, the dominant classifier, no. 5, also has error zero and covers the interval 0-50.

Raising the error threshold caused the approximation to deteriorate. In Figure 5, $\epsilon_0 = 500$. The prediction curve seems to ignore the break at $x = 50$, reflecting the change in slope only gradually. The classifier list showed several that bridged the break point, with predictates from about 30 to 70. Evidently, with a larger error threshold, the system was not forced, as in Figure 3, to evolve classifiers that corresponded closely to

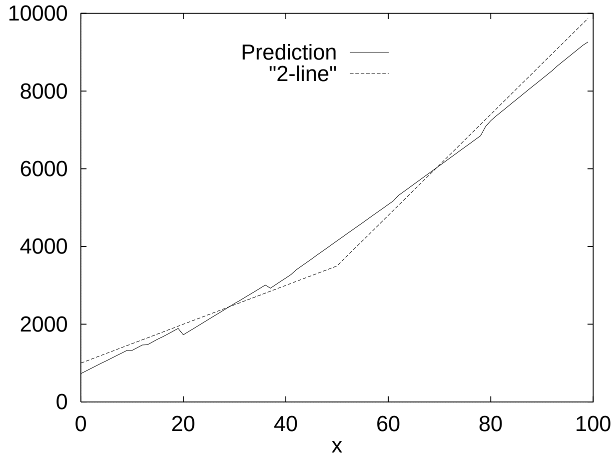

Figure 5: Approximation to "2-line" with $\epsilon_0 = 500$

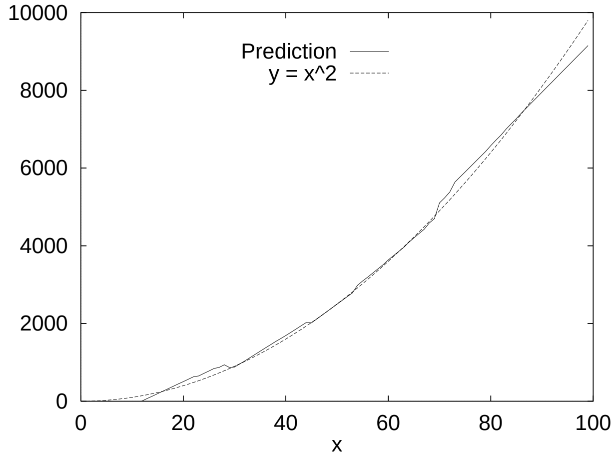

Figure 6: Piecewise-linear approximation to parabola with $\epsilon_0 = 500$.

the two function segments.

Figures 6 through 12 show typical results on parabola and sine functions. Parameters were the same as in the "2-line" experiments except $\eta = 0.2$ (an insignificant difference). Values for ϵ_0 are given in the captions. The two values chosen for each function are equivalent to 5% and 1% of the functions' ranges. To highlight the dominant classifiers and save space, the classifier lists include only the highest-fitness classifiers; the full populations are three or four times larger.

While the parabola figures show good linear approximations with a small number of classifiers, it is somewhat surprising that—unlike the piecewise-constant case—the sizes of the interval predicates do not seem to reflect the function's slope. Perhaps for piecewise-linear approximation a different analysis is in order: predicate length may be more related to curve straight-

[1]The prediction values represent the most recent weight-vector calculation and are not particularly significant. As in Section 3, this and all following experiments were run for 50,000 input cycles.

	CONDITION	PRED	ERR	FITN	NUM
0.	|.........OOOOOOOOOOO|	9375.	246.	.286	151
1.	|......OOOOOOOOOOOOOO|	9014.	431.	.122	54
2.	|.....oOOOOOOOOOOOOOO|	8985.	432.	.176	80
3.	|.....oOOOOOOOOOOOOOO|	8906.	456.	.037	37
4.	|....oOOOOOOOOOOOOOOO|	8744.	456.	.035	33
5.	|OOOOOOOOOOOOOOo.....|	4184.	379.	.077	35
6.	|OOOOOOOOOOOOOO......|	3756.	247.	.258	122
7.	|OOOOOOOOOOOOOo......|	3564.	256.	.066	32
8.	|OOOOOOOOOOOOo.......|	3193.	262.	.035	18
9.	|OOOOOOOOOOOo........|	2250.	204.	.211	125

Figure 7: High fitness classifiers from experiment of Figure 6.

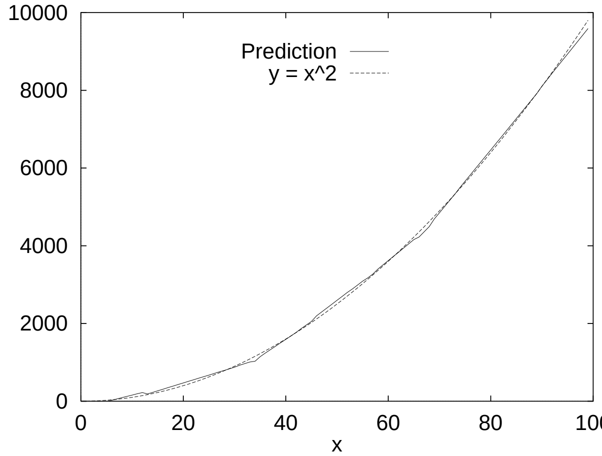

Figure 8: Piecewise-linear approximation to parabola with $\epsilon_0 = 100$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|............oOOOOOOO|	9591.	77.	.823	212
1.	|..........oOOOOOOO..|	7734.	75.	.081	21
2.	|.........oOOOOOOOO..|	7687.	89.	.035	9
3.	|......oOOOOOOo......|	4438.	79.	.731	184
4.	|......oOOOOo........|	3149.	26.	.123	41
5.	|..oOOOOOOo..........|	1868.	79.	.228	39
6.	|..oOOOOOo...........|	1621.	61.	.072	13
7.	|OOOOOOO.............|	967.	104.	.607	219
8.	|OOOOOOo.............|	749.	85.	.074	16

Figure 9: High fitness classifiers from experiment of Figure 8.

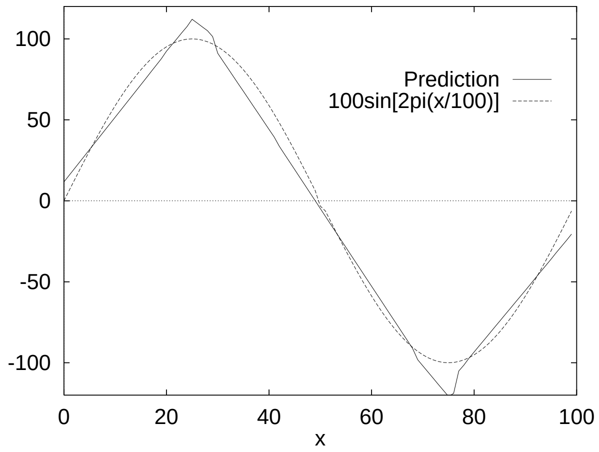

Figure 10: Piecewise-linear approximation to sine function with $\epsilon_0 = 10$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|OOOOOO..............|	120.	10.	.292	76
1.	|OOOOOo..............|	118.	10.	.083	22
2.	|OOOOO...............|	111.	7.	.572	158
3.	|....OOOOo...........|	66.	5.	.032	15
4.	|...............oOOOO|	-20.	4.	.792	173
5.	|.............oOOOOOO|	-23.	12.	.101	78
6.	|.....OOOOOOOOOOo....|	-130.	8.	.888	234

Figure 11: High fitness classifiers from experiment of Figure 10.

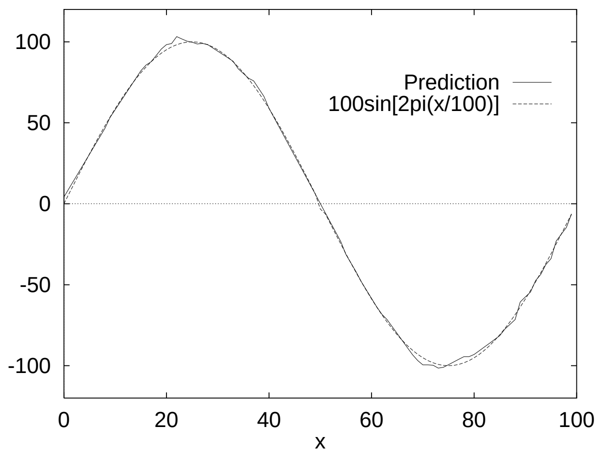

Figure 12: Piecewise-linear approximation to sine function with $\epsilon_0 = 2$.

ness than steepness. For the sinewaves, the approximation overshoots the peaks when ϵ_0 is large (Figure 10), but this effect disappears with smaller values and the curve is quite nicely matched (Figure 12). Figure 11 suggests that the system divides the approximation into classifiers for the beginning, middle, and end of the curve.

6 Multi-dimensional Input

XCSF was initially tested on functions of more than one variable by letting $y = f(x_1, ...x_n)$ be a linear function, i.e., a hyperplane function. The system rapidly evolved solutions with one or a few classifiers and arbitrary accuracy. This was expected, since the classifiers' weight vectors are effectively linear functions. To test XCSF on a multi-dimensional nonlinear function, we chose, somewhat arbitrarily, $y = [(x_1^2 + ... + x_n^2)/n]^{1/2}, 0 \leq x_i < 100$, a sort of "rms" function. Experiments with $n = 3$ went well, so $n = 6$ was tried. Parameters were the same as previously, except $N = 3200$, $\beta = 0.1$, $\theta_{GAsubs} = 200$, and $\eta = 1.0$. The error threshold $\epsilon_0 = 1$ (1% of the range). In contrast to previous experiments in which instances were chosen randomly from a fixed data set, instances were picked randomly from the domain. Figure 13 plots the system error and population size.

Starting initially very high, the system error (a moving average of the absolute difference between XCSF's prediction and the actual function value) fell rapidly to less than 1 (or .01 as plotted on this graph). The population size—in macroclassifiers—rose quickly to about 2400 and stayed there. So the system seemed to have little difficulty approximating the function to within 1%, though quite a few classifiers were required.

7 Classifier Architecture

In XCS, as in other classifier systems, the classifier prediction is a scalar, and the system adapts the classifier conditions and the prediction scalars to find accurate classifiers that are as general as possible. In XCSF, the prediction was replaced by a weight vector computing a linear function, leading to a more powerful and subtle co-adaptation of the condition and the prediction. As an extreme but instructive example, XCSF can approximate a very high-dimensional linear function with $O(1)$ classifiers, far less than required using scalar predictions.

It might be fruitful to extend the concept of classifier structure from the traditional <condition>:<action> ⇒ <prediction> to the more general <condition truth function>:<action range> ⇒ <prediction function>. The action range would be a finite interval of effector values, e.g., rudder angle. The condition would be a truth function of x and the prediction a function of x and a, where a is a value in the action range. The two functions and the range would co-adapt. Lanzi (1999) uses condition functions that are Lisp S-expressions. The present paper has investigated linear prediction functions. It is not hard to imagine a classifier that, for a given subspace of x and a finite range of a, predicts the payoff for each x, a combination. Such classifiers could be a step toward classifier systems capable of continuous actions. Even with fixed actions, the possibility of co-adapting condition and prediction functions should lead to more powerful—i.e., accurate and efficient—generalization in classifier systems.

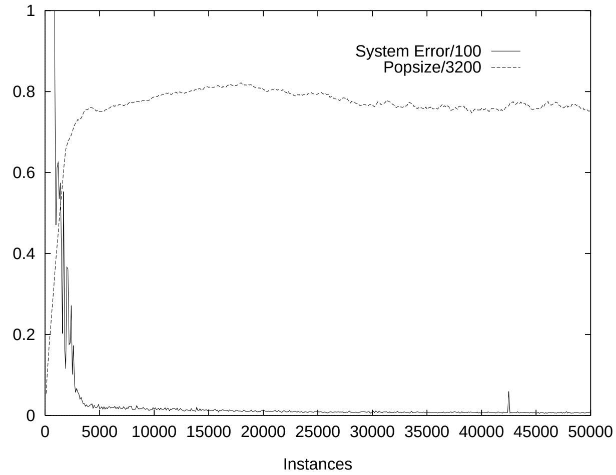

Figure 13: System error/100 and population size/3200 for approximation to six-dimensional "rms" function. $\epsilon_0 = 1$.

8 Summary and Conclusions

This paper introduced a classifier system, XCSF, designed to learn approximations to functions. The prediction estimation mechanism was used to form the approximations: given an input vector x, the value y of the function to be approximated was treated as a payoff to be learned. In its first incarnation, XCSF produced piecewise-constant approximations. A more advanced version added a weight vector to each classifier, permitting the approximation to be piecewise-linear. Tests on simple one-dimensional functions yielded arbitrarily close approximations, according to the setting of an error parameter. The system tended to evolve classifiers that distributed themselves reasonably efficiently over the function's domain, though some overlap occurred together with the presence of a moderate number of redundant low-fitness classifiers. In limited tests on a six-dimensional nonlinear function, XCSF rapidly formed highly accurate approximations, though the number of classifiers required was much larger than for the one-dimensional functions.

Future work should continue with multi-dimensional functions, to determine the technique's general viability and estimate its complexity in terms of learning time and resources (classifiers) required. Since XCSF approximates linear functions effortlessly, regardless of dimensionality, it is likely that the complexity will relate to the degree of "smoothness" or "flatness" in hyperspace that the function exhibits. Comparisons should be made with fuzzy classifier systems, which appear to be quite different in concept: the output of a fuzzy system is computed jointly by more than one classifier, whereas in XCSF an accurate output can in principal be computed by just one.

Function approximation with XCSF could be useful for on-line learning of any function or mapping from a vector of input values to a function or output value. An example would be financial time-series prediction, where a future price is presumably an approximable function of known prices or other quantities at earlier times in the series.

XCSF's approximation method can perhaps be extended to decision problems in which a decision, 1 or 0, depends on which side of a decision surface an input is on. XCSF would learn an approximation to the decision surface and then a given input's position relative to that surface could be determined.

Piecewise-linear function approximation in XCSF is based on the idea of *calculating* a classifier's prediction, and this leads to the concept of a *generalized classifier* in which the condition is a truth function of the input x and the prediction is a function of x and an action a. Such a classifier would apply in the subspace of the $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ mapping defined by the condition function and an action range specified in the classifier. The action range could be continuous, permitting selection of the best action value by maximizing the payoff over the range, a possible step toward classifier systems with continuous actions.

Acknowledgements

The author appreciates the comments of four anonymous reviewers, and has responded as space allowed. This work was partially supported by NuTech Solutions Inc.

References

Bonarini, A. (2000). An Introduction to Learning Fuzzy Classifier Systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson (Eds.), *Learning Classifier Systems. From Foundations to Applications*, Volume 1813 of *LNAI*, Berlin, pp. 83–104. Springer-Verlag.

Butz, M. V. and S. W. Wilson (2001). An Algorithmic Description of XCS. See Lanzi, Stolzmann, and Wilson (2001).

Lanzi, P. L. (1999). 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, and R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 345–352. Morgan Kaufmann: San Francisco, CA.

Lanzi, P. L., W. Stolzmann, and S. W. Wilson (Eds.) (2001). *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000).* Springer-Verlag.

Mitchell, T. M. (1997). *Machine Learning*. Boston, MA: WCB/McGraw Hill.

Valenzuela-Rendón, M. (1991). The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*, pp. 346–353.

Widrow, B. and M. E. Hoff (1988). Adaptive switching circuits. In J. A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research*, pp. 126–134. Cambridge, MA: The MIT Press.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation 3*(2), 149–175.

Wilson, S. W. (1998). Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 665–674. Morgan Kaufmann: San Francisco, CA.

Wilson, S. W. (2001). Mining Oblique Data with XCS. See Lanzi, Stolzmann, and Wilson (2001).