Generalization in the XCSF Classifier System: Analysis, Improvement, and Extension

Pier Luca Lanzi lanzi@elet.polimi.it
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Milano, I-20133, Italy

Daniele Loiacono loiacono@elet.polimi.it
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Milano, I-20133, Italy

Stewart W. Wilson wilson@prediction-dynamics.com
Prediction Dynamics, Concord, MA 01742, USA

David E. Goldberg deg@uiuc.edu
Department of General Engineering
University of Illinois, Urbana, IL 61801, USA

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

Abstract
We analyze generalization in XCSF and introduce three improvements. We begin by showing that the types of generalizations evolved by XCSF can be influenced by the input range. To explain these results we present a theoretical analysis of the convergence of classifier weights in XCSF which highlights a broader issue. In XCSF, because of the mathematical properties of the Widrow-Hoff update, the convergence of classifier weights in a given subspace can be slow when the spread of the eigenvalues of the autocorrelation matrix associated with each classifier is large. As a major consequence, the system's accuracy pressure may act before classifier weights are adequately updated, so that XCSF may evolve piecewise constant approximations, instead of the intended, and more efficient, piecewise linear ones. We propose three different ways to update classifier weights in XCSF so as to increase the generalization capabilities of XCSF: one based on a condition-based normalization of the inputs, one based on linear least squares, and one based on the recursive version of linear least squares. Through a series of experiments we show that while all three approaches significantly improve XCSF, least squares approaches appear to be best performing and most robust. Finally we show how XCSF can be extended to include polynomial approximations.

Keywords
Learning classifier systems, computed prediction, XCS.

1 Introduction

Learning classifier systems are a method of machine learning introduced almost 30 years ago by John H. Holland, the father of Genetic Algorithms (Holland 1975). In a learning classifier system, learning is viewed as a process of ongoing adaptation to an unknown environment which provides feedback in terms of numerical reward. The reward is used to guide the evolution of a set (termed *population*) of rules (*classifiers*) which represent the solutions to the target task.

Learning classifier systems employ (i) temporal difference learning (Sutton and Barto 1998) to distribute the incoming reward to the rules that are accountable for it, so as to determine the most promising rules to solve the target task; and (ii) genetic algorithms (Holland 1975; Goldberg 1989) to search the space of feasible rules so as to evolve "*the best*" (usually interpreted as *the most compact*) set of rules that solves the target task.

The XCS classifier system (Wilson 1995) represents an important milestone in learning classifier system research. XCS couples effective temporal difference learning, implemented as a modification of the well-known Q-learning (Watkins and Dayan 1992), to a niche genetic algorithm guided by an accuracy based fitness to evolve solutions made of accurate maximally general classifiers. XCS has been proved more effective than other models presented in the literature so far and it can be currently considered a major research direction in this area. In data-mining applications, XCS can perform better than some well-known traditional machine learning techniques (Wilson 2001b; Bernadó, Llorà, and Garrell 2001; Lanzi 2001; Dixon, Corne, and Oates 2002). However, successful applications in multistep problems are still restricted to small problems (Wilson 1995; Wilson 1998; Butz et al. 2003; Butz et al. 2005).

Recently, Wilson (2001a, 2002) introduced XCSF an extension of XCS in which the classifier prediction is no longer represented by a parameter but is computed as a linear combination of the classifier inputs and a weight vector maintained by each classifier. In all other models of learning classifier systems, the incoming reward is used to update a prediction (or strength) parameter. In XCSF gradient descent is used to update classifier weights based on (i) the difference between the current (computed) prediction and a target prediction value; (ii) current classifier inputs; and (iii) the current weight vector. As a result, XCSF evolves classifiers which represent piecewise linear approximations of parts of the reward surface associated with the problem solution. XCSF has been initially applied to approximate functions of one or more variables. The initial results discussed in (Wilson 2002; Wilson 2001a) show that XCSF can evolve populations of classifiers which accurately approximate the target function.

In this paper, we take the original work of Wilson (2001a, 2002) some steps further. We apply XCSF to approximate a simple function, the discrete sine, and show that while XCSF always evolves accurate approximations, the type of evolved generalizations depends on the input range. When the range of classifier inputs is limited to small values (as those used in the original paper), XCSF evolves *piecewise linear* approximations, as expected. When the range of classifier inputs includes large values, XCSF does not exploit the potential of linear approximation and tends to evolve accurate *piecewise constant* approximations of the target function, similar to those evolved by earlier models (Wilson 1995; Wilson 2001b).

Through a theoretical analysis and known results from the literature on linear approximators (Haykin 1986), we show that the observed behavior is due to a more general issue regarding the mathematical properties of the Widrow-Hoff update (Widrow and Hoff 1988) that, in XCSF is used to update classifier weights. In fact, Widrow-Hoff updates can result in slow convergence in some area of the weight space when the distribution of the inputs is such that there is a large spread between the smallest and the largest eigenvalues of the autocorrelation matrix of the inputs (Haykin 1986). We show that, when it comes to functions of one variable such as those used in our initial experiments, such spread is *also* affected by the range of classifier inputs and it is usually wider when the range of classifier inputs includes large values. Accordingly, when the range of classifier inputs includes large values, the classifier weights may adapt slowly. As a result, the system's accuracy pressure may act and cause classifier reproduction before the weights are adequately updated. Thus, XCSF may evolve accurate classifiers

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

which however do not fully exploit the generalization capabilities that the linear computation of the classifier prediction would allow. In effect, when certain conditions over the distribution of classifier inputs hold, so that the convergence of the Widrow-Hoff is extremely slow, XCSF tends to evolve *piecewise constant* approximations similar to those that would be possible with XCSI (Wilson 2001b), instead of exploiting the full capabilities of its representation.

While our analysis was performed to explain a rather simple effect of classifier input ranges on the generalization capabilities of XCSF, the effect we discovered is due to a more general issue. Overall, our analysis suggests that as long as we use Widrow-Hoff to update classifier weights in XCSF the convergence to the optimal weight vectors may be extremely slow when the distribution of classifier inputs is such that there is a large spread in the eigenvalues of the autocorrelation matrix. When this happens, piecewise linear approximation in XCSF becomes less effective since some parts of the weight space will be prone to slower convergence eventually making the best generalizations almost unreachable.

To improve the generalization capabilities of XCSF, so as to make piecewise linear approximation effective on any input range, we propose three methods. The first one, very simple but limited in generality, consists of keeping the Widrow-Hoff update while normalizing the inputs before they are actually used to compute and to update the classifier prediction. This very simple solution can reduce the impact of input range over the spread of the eigenvalues of the autocorrelation matrix, though it does not solve the more general issue which affects the Widrow-Hoff update. The second one, with more theoretical guarantees than Widrow-Hoff, consists of replacing the original Widrow-Hoff update with *linear least squares* (Haykin 1998). The third one is the recursive version of the linear least squares. We compare the original XCSF with the three proposed extensions on different problems showing that (i) all the three approaches can lead to more effective generalization, and that (ii) least squares approaches appear to be more robust and best performing. Finally, we extend XCSF to allow polynomial approximations. Through a number of experiments we show that XCSF can exploit the polynomial basis and evolve more compact solutions than is possible with linear approximations.

2 Description of XCSF

XCSF (Wilson 2001a; 2002) extends the typical concept of classifiers through the introduction of a computed classifier prediction. To develop XCSF, XCS has to be modified in three respects: (i) classifier conditions are extended for numerical inputs; (ii) classifiers are extended with a vector of weights $\vec{w}$, that is used to compute the classifier prediction; finally, (iii) the original update of the prediction parameter must be modified so that the weights are updated instead of the prediction parameter. These three modifications result in a version of XCS, XCSF (Wilson 2001a; 2002), that maps numerical inputs into actions (Wilson 2004) with an associated calculated prediction.

2.1 Classifiers

In XCSF, classifiers consist of a condition, an action, and four main parameters. The condition specifies which input states the classifier matches; as in XCSI (Wilson 2001b), it is represented by a concatenation of interval predicates, $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are integers, though they might be also real. The action

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

specifies the action for which the payoff is predicted.[1] The four parameters are: the weight vector $\vec{w}$, used to compute the classifier prediction as a function of the current input; the prediction error ε, that estimates the error affecting the classifier prediction; the fitness F that estimates the accuracy of the classifier prediction; and the numerosity *num*, a counter used to represent different copies of the same classifier. The weight vector $\vec{w}$ has one weight w_i for each possible input, and an additional weight w_0 corresponding to a constant input x_0, that is set as a parameter of XCSF.

2.2 Performance Component

XCSF works very similarly to XCS. At each time step t, XCSF builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input s_t; if [M] contains less than θ_{mna} actions, *covering* takes place and creates a new classifier that matches the current inputs and has a random action. Each interval predicate $int_i = (l_i, u_i)$ in the condition of a covering classifier is generated as $l_i = s_t(i) - rand_1(r_0)$, and $u_i = s_t(i) + rand_1(r_0)$, where $s_t(i)$ is the input value of state s_t matched by the interval predicated int_i, and the function $rand_1(r_0)$ generates a random integer in the interval $[0, r_0]$ with r_0 fixed integer. The weight vector $\vec{w}$ of covering classifiers is initialized with zero values (note in the original papers, weights are initialized with values in [-1,1]); all the other parameters are initialized as in XCS (see Butz and Wilson (2002)).

For each action a_i in [M], XCSF computes the *system prediction* which estimates the payoff that XCSF expects when action a_i is performed. As in XCS, in XCSF the *system prediction* of action a is computed by the fitness-weighted average of all matching classifiers that specify action a. However, in contrast with XCS, in XCSF the classifier prediction is computed as a function of the current state s_t and the classifier vector weight $\vec{w}$. Accordingly, in XCSF the system prediction is a function of both the current state s and the action a. Following a notation similar to (Butz and Wilson 2001), the system prediction for action a in state s_t, $P(s_t, a)$, is defined as

$$P(s_t, a) = \frac{\sum_{cl \in [M]|_a} cl.p(s_t) \times cl.F}{\sum_{cl \in [M]|_a} cl.F} \tag{1}$$

where *cl* is a classifier, $[M]|_a$ represents the subset of classifiers in [M] with action a, *cl.F* is the fitness of *cl*; $cl.p(s_t)$ is the prediction of *cl* computed in the state s_t. In particular, $cl.p(s_t)$ is computed as

$$cl.p(s_t) = cl.w_0 \times x_0 + \sum_{i>0} cl.w_i \times s_t(i)$$

where $cl.w_i$ is the weight w_i of *cl* and x_0 is a constant input. The values of $P(s_t, a)$ form the *prediction array*. Next, XCSF selects an action to perform. The classifiers in [M] that advocate the selected action are put in the current *action set* [A]; the selected action is sent to the environment and a reward r is returned to the system together with the next input state s_{t+1}. Note that when XCSF is applied to function approximation problems, as in this paper, there is only one (dummy) action that the system can perform; it has no actual effect on the environment (Wilson 2001a, 2002).

[1] In (Wilson 2001a; 2002), XCSF classifiers did not have an action; it was added later in (Wilson 2004). Note however that the two versions presented in (Wilson 2001a; 2002) and (Wilson 2004) are identical except for the introduction of classifier actions.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

2.3 Reinforcement Component

XCSF uses the incoming reward r to update the parameters of classifiers in action set [A]. First, the reward r is used to update the weight vector $\vec{w}$ using a *modified delta rule* as follows (Widrow and Hoff 1988; Wilson 2002). For each classifier $cl \in [A]$, each weight $cl.w_i$ is adjusted by a quantity Δw_i computed as

$$\Delta w_i = \frac{\eta}{|\vec{x}_{t-1}|^2}(r - cl.p(s_{t-1}))x_{t-1}(i) \tag{2}$$

where η is the correction rate and $\vec{x}_{t-1}$ is defined as the input state vector s_{t-1} augmented by a constant x_0 (i.e. $\vec{x}_{t-1} = \langle x_0, s_{t-1}(1), s_{t-1}(2), \ldots, s_{t-1}(n) \rangle$) and $|\vec{x}_{t-1}|^2$ is the norm of vector $\vec{x}_{t-1}$; for further details refer to (Wilson 2002). The values Δw_i are used to update the weights of classifier *cl* as:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \tag{3}$$

Then the prediction error ε is updated as

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|r - cl.p(s_{t-1})| - cl.\varepsilon).$$

Classifier fitness is updated as in XCS. First, the *raw accuracy* κ of the classifiers in [A] is computed as follows:

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon \le \varepsilon_0 \\ \alpha(\varepsilon/\varepsilon_0)^{-\nu} & \text{otherwise.} \end{cases} \tag{4}$$

The *raw accuracy* κ is used to calculate the *relative accuracy* κ' as

$$\kappa' = \frac{(\kappa \times num)}{\sum_{cl \in [A]}(cl.\kappa \times cl.num)}. \tag{5}$$

where $cl.\kappa$ is the *raw accuracy* of classifier *cl*, as computed in Equation 4; *cl.num* is the *numerosity* of classifier *cl*. Finally the *relative accuracy* κ' is used to update the classifier fitness as: $F \leftarrow F + \beta(\kappa' - F)$. An algorithmic description of the overall update procedure is reported as Algorithm 1, while the prediction update for one classifier is reported as Algorithm 2.

2.4 Discovery Component

In XCSF, the discovery component works as in XCSI (Wilson 1995; 2001b). A genetic algorithm (GA) (Holland 1975) is applied to the classifiers in the current action set [A] if the average time since the last GA application to the classifiers in [A] exceeds a threshold θ_{ga}. Two offspring classifiers are generated by reproducing, crossing, and mutating the parents. The offspring are inserted into the population. As happens in all the other models of classifier systems, parents stay in the population competing with their offspring. The genetic operators, crossover, mutation, and subsumption work as in XCSI (Wilson 2001b).

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

Algorithm 1 XCSF: Update the classifiers in [A]

```
1: procedure UPDATE_SET([A], $s$, $P$, [P]) ▷ $P$ is the target payoff, $s$ is the current input
2:     ass ← 0;                                          ▷ Compute action set size
3:     for all cl in [A] do
4:         ass ← ass + cl.num;
5:     end for
6:     for all cl∈ [A] do
7:         cl.exp++;
8:         $\hat{p}$ ← cl.p(s);                            ▷ Compute classifier prediction
9:         UPDATE_PREDICTION(cl, $s$, $P$);               ▷ Update prediction of cl
10:        if cl.exp $< 1/\beta$ then                        ▷ Update prediction error
11:            cl.$\varepsilon$ ← cl.$\varepsilon$ + $(|P - \hat{p}|$ − cl.$\varepsilon)$/cl.exp;
12:        else
13:            cl.$\varepsilon$ ← cl.$\varepsilon$ + $\beta(|P - \hat{p}|$ − cl.$\varepsilon)$;
14:        end if
15:        if cl.exp $< 1/\beta$ then                        ▷ Update action set size
16:            cl.as ← cl.as + (ass − cl.as)/cl.exp;
17:        else
18:            cl.as ← cl.as + $\beta$(ass − cl.as);
19:        end if
20:    end for
21:    UPDATE_FITNESS([A]);
22:    if doActionSetSubsumption then
23:        DO_AS_SUBSUMPTION([A],[P]);
24:    end if
25: end procedure
```

3 Design of Experiments

All the experiments discussed in this paper involve single step problems and are performed following the standard design used in the literature (Wilson 1995; 2002). In each experiment XCSF has to learn to approximate a target function $f(x)$; each experiment consists of a number of problems that XCSF must solve. For each problem, an example $\langle x, f(x)\rangle$ of the target function $f(x)$ is randomly selected; x is input to XCSF, which computes the approximated value $\hat{f}(x)$ as the expected payoff of the only available dummy action; the action is virtually performed (the action has no actual effect), and XCSF receives a reward equal to $f(x)$. XCSF learns to approximate the target function $f(x)$ by evolving a mapping from the inputs to the payoff of the only available action. Each problem is either a *learning* problem or a *test* problem. In *learning* problems, the genetic algorithm is enabled; during *test* problems it is turned off. Classifier parameters are always updated. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate.

XCSF performance is measured as the accuracy of the evolved approximation $\hat{f}(x)$ with respect to the target function $f(x)$. To evaluate the evolved approximation $\hat{f}(x)$ we measure the *mean absolute error* (MAE) and the *mean square error* (MSE) defined as

$$MAE = \frac{1}{n}\sum_x |f(x) - \hat{f}(x)|, \qquad MSE = \frac{1}{n}\sum_x (f(x) - \hat{f}(x))^2,$$

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

Algorithm 2 XCSF: Update the prediction of *cl* with the modified delta rule.

```
1:  procedure UPDATE_PREDICTION(cl, $s$, $P$)
2:      error $\leftarrow P -$ cl.$p(s)$;
3:      norm $\leftarrow x_0^2$;                                ▷ Compute $|x|^2$
4:      for $i \in \{1, \ldots, |s|\}$ do
5:          norm $\leftarrow$ norm $+s(i)^2$;
6:      end for
7:      correction $\leftarrow \frac{\eta}{norm} \times$ error;          ▷ Compute the overall correction
8:      cl.$w_0 \leftarrow x_0 \times$ correction;     ▷ Update the weights according to the correction
9:      for $i \in \{1, \ldots, |s|\}$ do
10:         cl.$w_i \leftarrow$ cl.$w_i + s(i) \times$ correction;
11:     end for
12: end procedure
```

where n is the number of points for which $f(x)$ is defined. In particular we use as estimates of the expected values of MAE and MSE, the average MAE and MSE over the performed experiments $\overline{MAE}$ and $\overline{MSE}$. All the statistics reported in this paper are averaged over 50 experiments. All the experiments reported have been conducted on `xcslib` (Lanzi 2002). The implementation of XCSF used in this work was previously validated by duplicating Wilson's original results (Wilson 2004), see (Lanzi et al. 2005b) for details.

4 Experiments on XCSF Sensitivity to Input Range

In this section, we investigate the sensitivity of XCSF with respect to the input domain and we analyze how the range of input values impacts the type of approximation evolved by XCSF. For this purpose, we perform a very simple experiment and we apply XCSF to approximate the discrete sine $f_{ds}(x)$ (Wilson 2001a),

$$f_{ds}(x) \quad = \quad 100 \times sin(\frac{2\pi x}{100}) \tag{6}$$

in two different input ranges, $I_1 = [0, 100]$ and $I_2 = [1000, 1100]$. The parameters for the two experiments are set as follows: $N = 800$; $\eta = 0.2$; $\beta = 0.2$; $\alpha = 0.1$; $\epsilon_0 = 10$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{nma} = 1$, $\theta_{del} = 50$; $\theta_{GA} = 50$; $\delta = 0.1$; GA-subsumption is on with $\theta_{sub} = 50$; while action-set subsumption is off; the parameters for integer conditions are $m_0 = 20$, $r_0 = 10$ (Wilson 2001b); each experiment consists of 50000 learning problems. As discussed by Wilson (2002), in the Widrow-Hoff update the value of x_0 must be selected appropriately within the input range (Widrow and Hoff 1988). Accordingly, we set $x_0 = 50$ when $x \in I_1$ and $x_0 = 1050$ when $x \in I_2$.

The discrete sine is periodic and, apart from x_0, none of XCSF parameters depends on the input values, therefore, we should expect that XCSF performs the same in the two cases.

Figure 1a and Figure 1b compare the target function $f_{ds}(x)$ (dashed line) to the approximation $\hat{f}_{ds}(x)$ (solid line) evolved by XCSF respectively when $x \in I_1$ (Figure 1a) and $x \in I_2$ (Figure 1b); curves are averages over 50 experiments; vertical bars represent the variance over the 50 experiments. In both cases, the solutions evolved by XCSF are accurate. In the former case ($x \in I_1$) the average absolute error $\overline{MAE} \pm \sigma$ is 5.65 ± 0.67 (corresponding to a $\overline{MSE} \pm \sigma$ of 46.63 ± 10.48), while the worst solution has an absolute error of 7.18, i.e., in all the experiments the average error is below the ϵ_0 threshold. In

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

the latter case ($x \in I_2$) the average absolute error $\overline{MAE} \pm \sigma$ is 5.63 ± 0.41 (corresponding to a $\overline{MSE}$ of 47.68 ± 7.09), while the worst solution has an absolute error of 7.9, i.e., also in this case in all the experiments the average error is below the ϵ_0 threshold.

Nevertheless, the approximations evolved by XCSF in the two cases differ. In particular, when the input values are larger, that is in I_2, XCSF evolves solutions that on average fit poorly the convex and concave regions, and overall have higher variance. To quantify such difference and to test whether it is statistically significant we apply a Student's t-test to compare the distributions of MSE values in the two cases and the parametric F-test to compare their variances (Conover 1998). We find that the difference in $\overline{MSE}$ variances is significant with a confidence level of 99.7%; in contrast, the difference in the averages is not significant.

The most important difference in the solutions evolved in the two input ranges can be detected only by analyzing the single runs. In Figure 1c and Figure 1d we report two typical solutions evolved by XCSF for the two input ranges. As can be noted, when $x \in I_1$, the approximation evolved (Figure 1c) is clearly piecewise linear and follows the sine shape. In contrast, when $x \in I_2$, XCSF has evolved a piecewise constant approximation (Figure 1d), a generalization that is typical of XCSI (Wilson 2001b; Wilson 2002). Note also, that even when $x \in I_1$, the type of generalization tends to become piecewise constant for $x \in [75, 100]$ (Figure 1c). To show this difference more clearly, in Figure 2 we compare the distribution of the weights w_1 for the classifiers evolved in I_1 (Figure 2 upper plot) and in I_2 (Figure 2 lower plot). We consider the evolved values of w_1 since they determine the slope of the approximation evolved by each classifier. As can be noticed, in I_1 the values of w_1 are mainly grouped around three values two of which, that around -5 and that around 4, correspond to the slopes that approximate the three main curves present in the discrete sine in $[0, 25]$, $[25, 75]$, and $[75, 100]$. In contrast, in I_2 all the weights w_1 are between -0.1 and 0.1 so that all the classifiers approximate a horizontal line. As an example, Figure 3 reports the two populations corresponding to the approximation in Figure 1c and Figure 1d where each classifier is depicted by the segment it represents. As expected, in I_1 XCSF has evolved classifiers representing oblique segments, while in I_2, XCSF has evolved only classifiers representing horizontal segments.

This also has an effect on the generality of the evolved solutions: when applied to I_2, XCSF does not exploit the oblique parts of the target function: accordingly, the solutions evolved are more specific than those evolved for I_1. In fact, if we compare the distribution of classifier generality in the two cases (computed as the number of inputs matched) we have that when $x \in I_1$, the average classifier generality is 22.6 (that is, in I_1 classifier conditions on the average match 22.6 inputs), while when $x \in I_2$, the average classifier generality is 9.0 (that is, in I_1 classifier conditions on the average match 9.0 inputs). According to Student's t-test, this difference is statistically significant with a confidence level of 99.999%.

It is important to note that both the solutions evolved by XCSF in I_1 and I_2 are *actually accurate*, with respect to the error threshold ϵ_0, but qualitatively different. In I_1, XCSF can exploit its capability to represent linear approximations, in I_2, XCSF cannot.

The experiments on the discrete sine suggest that the effectiveness of XCSF's piecewise linear approximation is dramatically influenced by the input range. Such behavior is quite general and it was observed in other test problems (not reported here). Overall, it appears that in XCSF linear approximation is less effective in problems involving large input values. Apart from x_0, in XCSF only the classifier prediction and the weight update depend on the input range. Accordingly, in the next section, we analyze the

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

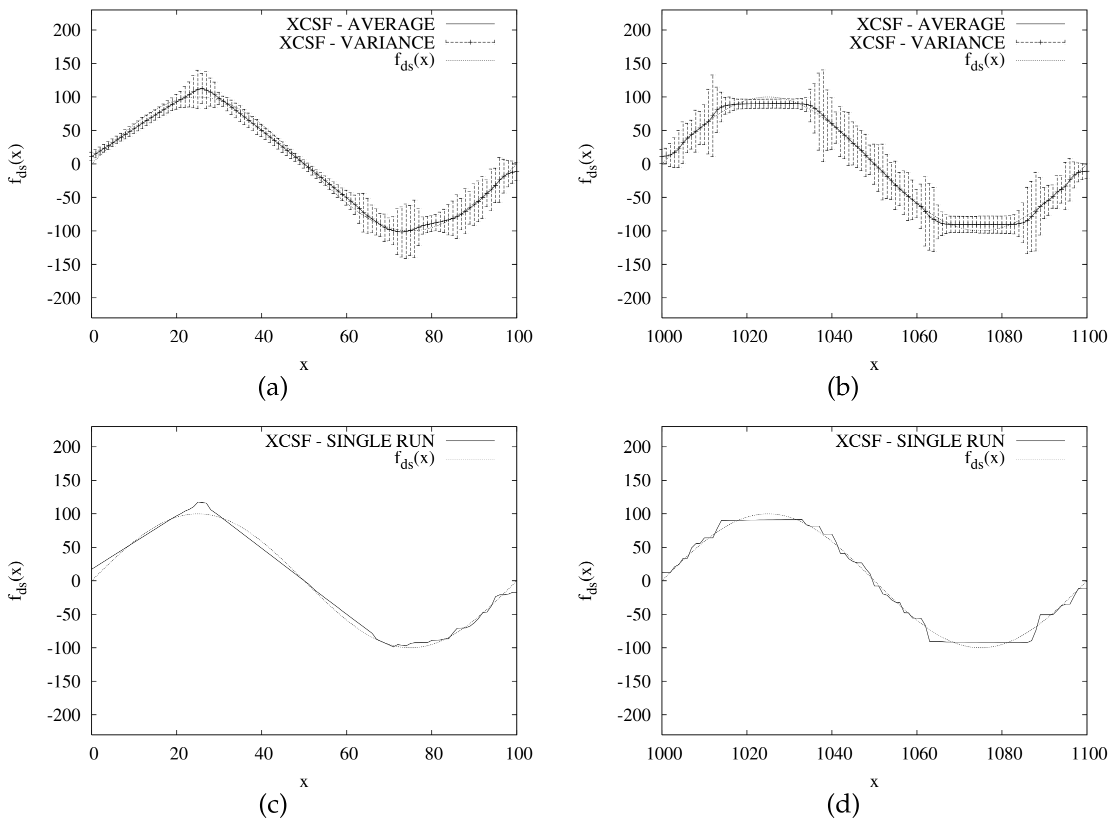


Figure 1: XCSF applied to the discrete sine with a population size $N = 800$ and an error threshold $\epsilon_0 = 10$. Comparison of XCSF approximation (solid line) against the target function (dashed line) when $x \in [0, 100]$, (a) reports the average over 50 runs, (c) reports a typical run; when $x \in [1000, 1100]$, (b) reports the average over 50 runs, (d) reports a typical run.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

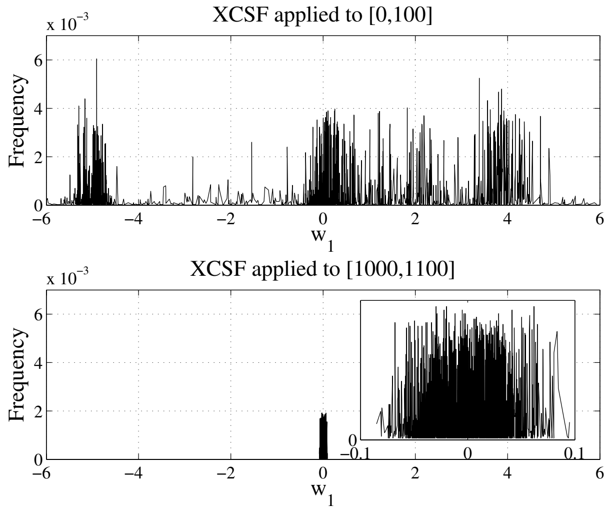


Figure 2: XCSF applied to the discrete sine. Distribution of weight w_1 for I_1 (upper plot) and for I_2 (lower plot).

convergence of the Widrow-Hoff delta rule and its effect on the update of the classifier prediction in XCSF so as to explain the results discussed here.

5 Convergence of Classifier Predictions

We now analyze from a theoretical standpoint the convergence of classifier weights in XCSF so as to explain the results discussed in the previous section.

5.1 Basic Definitions

We begin with some basic definitions. Suppose, we want to approximate a function $f(\vec{x})$ with XCSF. We focus on a subspace I of the function domain, and consider a classifier *cl* with weight $\vec{w}$, defined over I, that is, matching the elements in I. We define the error $J(\vec{w})$ associated to the weight vector $\vec{w}$ as

$$J(\vec{w}) = \frac{1}{|I|}\sum_{\vec{x}\in I}\left(f(\vec{x}) - \vec{w}\vec{x}\right)^2 \tag{7}$$

where $|I|$ is the number of problem points that belongs to I. Function $J(\vec{w})$ provides an absolute measure of the classifier approximation error, but what we really need to evaluate the goodness of a classifier is a measure of its approximation relative to the best possible linear approximation. Thus we define J^* as the minimum error associated to a linear approximation on I, that is,

$$J^* = \min_{\vec{w}} J(\vec{w}), \tag{8}$$

and the weight vector $\vec{w}^*$ associated to J^* as

$$\vec{w}^* = \arg\min_{\vec{w}} J(\vec{w}). \tag{9}$$

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

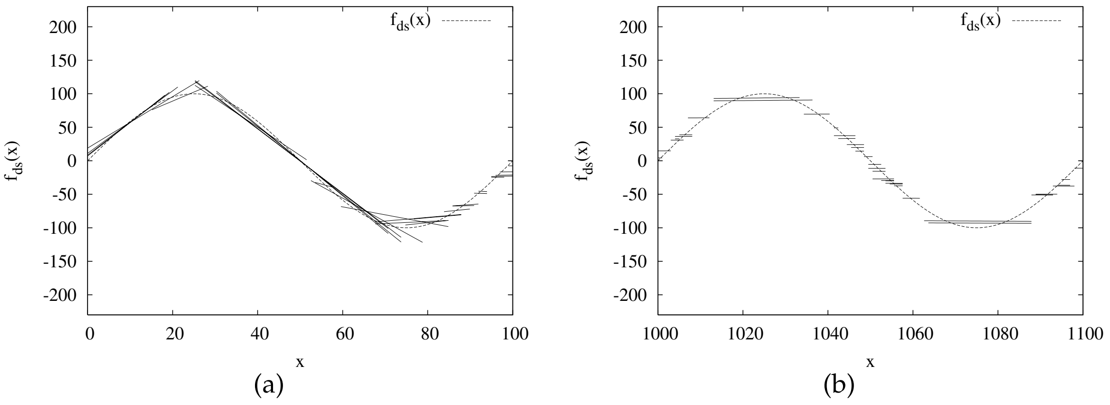


Figure 3: XCSF applied to the discrete sine: (a) population evolved in I_1 and (b) population evolved in I_2. The corresponding approximations are reported in Figure 1c and Figure 1d respectively.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

We can now introduce the (relative) error function $\tilde{J}(\vec{w})$, defined as

$$\tilde{J}(\vec{w}) = J(\vec{w}) - J^*, \tag{10}$$

and we can use $\tilde{J}$ to study the convergence of the classifier predictions by studying the convergence of $\tilde{J}(\vec{w}_t)$ for $t \to \infty$.

5.2 Theoretical Properties

To analyze the converge of the classifier prediction in XCSF we use some known results regarding the convergence of the modified Widrow-Hoff update used in XCSF (Haykin 1986). Consider a classifier *cl* that has been applied to the sequence of inputs $\vec{x_1} \ldots \vec{x_t}$. We define the matrix X_t of all the inputs, until time t, as

$$X_t = \begin{bmatrix} \vec{x_1} \\ \vdots \\ \vec{x_i} \\ \vdots \\ \vec{x_t} \end{bmatrix} \tag{11}$$

where $\vec{x_i}$ is the i-th input that the classifier has been applied to, and t is the total number of updates so far. Given X_t we define the autocorrelation matrix of the inputs, $R_{xx}(t)$, as

$$R_{xx}(t) = \frac{1}{t} X_t^T X_t. \tag{12}$$

To define the asymptotic behavior of $R_{xx}(t)$, we introduce R^*, defined as

$$R^* = \lim_{t \to \infty} R_{xx}(t). \tag{13}$$

(Haykin 1986) proves that the convergence property of Widrow-Hoff update depends on the eigenvalues of R^*. In particular, it shows that the expected value of the error

$\tilde{J}$, $E[\tilde{J}(w_t)]$, converges to a steady state if and only if the learning rate (η) satisfies the condition

$$0 < \frac{\eta}{E\left[\|\vec{x}\|^2\right]} < \frac{2}{\sum_{i=1}^{|\vec{x}|} \lambda_i} \tag{14}$$

where λ_i are the eigenvalues of R^* and $|\vec{x}|$ is the dimension of the input vector $\vec{x}$. In its simplified form, Equation 14 states that the learning rate η of the modified delta rule must be between 0 and 2. More generally, it also shows that in the modified delta rule, to guarantee the convergence, the choice of learning rate can be made without considering either the input range or the input distribution (Haykin 1986).

Each eigenvalue λ_i of R^* is bound to a linear combination of weights of $\vec{w}$, that is, each eigenvalue is bound to a section of the weight space; such combination is determined by the eigenvectors of R^*. (Haykin 1986) demonstrates that the convergence of the weight vector $\vec{w}_t$ to the optimal $\vec{w}^*$ is bound to the convergence to *zero* of the vector $\vec{v}$ defined as

$$\vec{v}(t) = Q^T \times (\vec{w}_t - \vec{w}^*)$$

where Q is the eigenvector matrix (Haykin 1986). More precisely, the convergence of each component of $\vec{v}$ is determined by the difference equation

$$\vec{v}_i(t+1) = (1 - \eta\lambda_i)\vec{v}_i(t) \tag{15}$$

where, η is the usual learning rate. From Equation 15 it is clear that the convergence of $\vec{v}$, and thus the convergence of $\vec{w}$ is related to the eigenvalues of R^*. In particular, a small eigenvalue λ_i causes a slow convergence of the corresponding component of $\vec{v}$. Thus since each component in $\vec{v}$ corresponds to a subspace of the weight space (determined by the corresponding eigenvector), a small eigenvalue causes a slow convergence of the weight values in the subspace bound to λ_i. On the other hand, the contribution to the overall error of such slowly converging weights will be much less than that of other weights associated to large eigenvalues of R^*. Overall, the time required by $E[\vec{w}]$ to converge to the optimum $\vec{w}^*$, is limited by the smallest eigenvalue. However, the error $E[\tilde{J}(w)]$ will not be affected much by the smallest eigenvalue of R^*. In fact, the expected value of error $\tilde{J}$, $E[\tilde{J}(w_t)]$, converges to a biased minimum $J^* \times M$. The term M is called *misadjustment*. When the eigenvalues of R_{xx} are widely spread, misadjustment is limited from the larger eigenvalue and it is bound by the learning rate η and on $E\left[\|\vec{x}\|^2\right]$. (Haykin 1986) shows that when the eigenvalues of the R^* matrix are widely spread, the misadjustment is primarily determined by the largest eigenvalues, whereas the average time required by the weight to converge is limited by the smallest eigenvalue of R^*. Thus, overall, Haykin (1986) shows that when the eigenvalue spread is large, the Widrow Hoff update can be very slow in some section of the weight subspace so that it requires a large number of iterations to converge.

5.3 Convergence of Classifier Weights in XCSF

The analysis we presented in the previous section suggests that when the distribution of classifier inputs in XCSF corresponds to a large spread in the eigenvalues of the autocorrelation matrix R^*, we should expect slow convergence of classifier weights in some area of the weight space. The results for the discrete sine between 1000 and 1100 discussed in Section 4 seem to support this. In [1000,1100], the weight w_1 of classifiers does not converge to those values which would allow XCSF to evolve classifiers representing the major piecewise linear approximations for the sine function–covering the slopes in the sine curve. From our analysis, we should expect that classifiers covering

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

cl	I	x_0	λ_m	λ_M	λ_M/λ_m	$\tilde{J}$
cl_l	$[0, 25]$	50	4.896×10^1	2.659×10^3	5.431×10^1	5.147×10^0
cl_L	$[1000, 1025]$	1050	2.699×10^1	2.128×10^6	7.884×10^4	1.045×10^3
cl_m	$(25, 75]$	50	1.020×10^2	5.106×10^3	5.006×10^1	9.803×10^0
cl_M	$(1025, 1075]$	1050	1.042×10^2	2.205×10^6	2.117×10^4	4.545×10^3
cl_u	$(75, 100]$	50	1.277×10^1	1.020×10^4	7.983×10^2	2.448×10^1
cl_U	$(1075, 1100]$	1050	2.513×10^1	2.285×10^6	9.094×10^4	1.033×10^3

Table 1: Eigenvalues of R^* for cl_l, cl_m, cl_u, cl_L, cl_M, and cl_U: *cl* identifies the classifier; I is interval matched by the classifier; x_0 is the value of the corresponding parameter in XCSF; λ_m is the minimum eigenvalue of R^*; λ_M is the maximum eigenvalue of R^*; λ_M/λ_m is the spread of the eigenvalues of R^*; $\tilde{J}$ is the error associated to the weight vector after 30000 updates.

such areas of the sine curve correspond to large spreads in the eigenvalues of R^*. We now go back to the discrete sine and analyze such eigenvalue spreads.

The asymptotic value of the autocorrelation matrix R^*, in the case of the approximation of a monodimensional function, is

$$R^* = \lim_{t\to\infty} R_{xx}(t) = E\left[\begin{pmatrix} x_0^2 & x_0x_1 \\ x_0x_1 & x_1^2 \end{pmatrix}\right]. \tag{16}$$

Given a classifier *cl* defined over an interval $I = [a, b]$, if we assume that the input values are uniformly distributed, we have that:

$$R^* = \begin{pmatrix} x_0^2 & x_0\frac{a+b}{2} \\ x_0\frac{a+b}{2} & \frac{a^2+ab+b^2}{3} \end{pmatrix}. \tag{17}$$

This equation does not allow us to derive a simple relation between its eigenvalues and classifier parameters. However, it is possible to devise a simple experiment to analyze the convergence speed of classifiers for the discrete sine with respect to the spread of the eigenvalues of R^*. For this purpose, we consider three classifiers, cl_l, cl_m, and cl_u, for the discrete sine between 0 and 100, matching respectively the intervals $I_l = [0, 25]$, $I_m = (25, 75]$, and $I_u = (75, 100]$. We compare the convergence speed of these classifiers, with the convergence speed of the corresponding classifiers for the discrete sine between 1000 and 1100, cl_L, cl_M, and cl_U, matching the intervals $I_L = [1000, 1025]$, $I_M = (1025, 1075]$, and $I_U = (1075, 1100]$. Table 1 reports the spread of each classifier. As can be noted the classifiers applying in $[1000, 1100]$ have large spreads (column λ_M/λ_m), much larger than those of the classifiers applying in $[0, 100]$. In addition, the spread of eigenvalues in $(75, 100]$ is larger than in $[0, 25]$ and $(25, 75]$, suggesting an explanation of why even in the interval $[0, 100]$ the generalizations evolved by XCSF would tend to be piecewise constant in $(75, 100]$. From the properties discussed in the previous section, we should expect a long time to converge for the classifiers in $[1000, 1100]$. To verify this, we perform 50 experiments; for each experiment we generate 30000 inputs from $[0, 100]$ and 30000 inputs from $[1000, 1100]$ which we use to train the two sets of classifiers. In this case, we do not use XCSF but just update the classifier predictions. In column $\tilde{J}$ of Table 1, for each classifier, we report the average error $\tilde{J}(\vec{w})$ over the 50 experiments at the end of the 30000 updates. The error $\tilde{J}(\vec{w})$ of classifiers matching inputs in $[0, 100]$ is much smaller than that of classifiers matching inputs in $[1000, 1100]$. Figure 4 reports the error $\tilde{J}(\vec{w})$ for the classifiers applying in

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

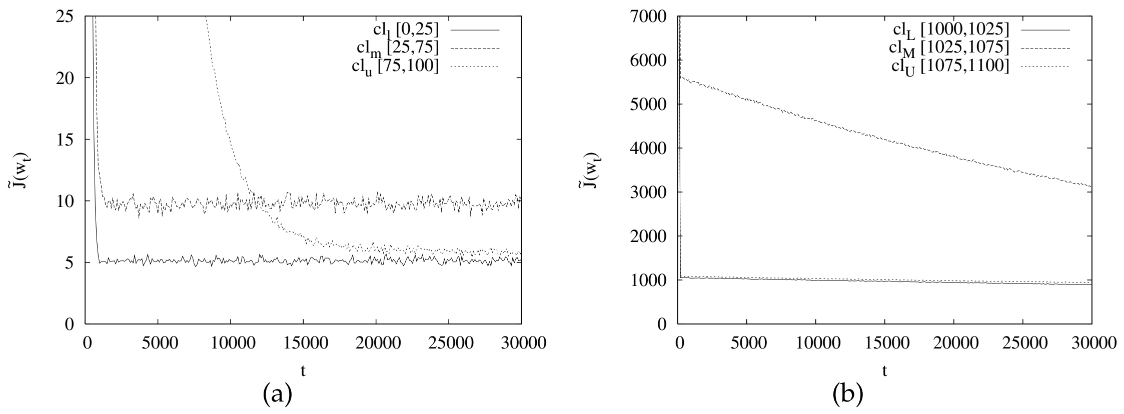


Figure 4: Convergence of the classifier predictions for (a) classifiers matching $I_1 = [0, 100]$ and for (b) classifiers matching $I_2 = [1000, 1100]$. Plots are averages over 50 runs. Note the difference in scale for the ordinate in the two plots.

$[0, 100]$ (Figure 4a) and for the classifiers applying in $[1000, 1100]$ (Figure 4b). As can be noted, classifiers cl_l and cl_m converge almost immediately to the optimal steady state value (Figure 4a), while cl_u converges slower, as early suggested by the larger eigenvalues spread. In contrast, all the classifiers applying in $[1000, 1100]$ have a very slow convergence (Figure 4b).

While the results discussed here support the analysis we presented in the previous section, we also analyzed how XCSF evolved accurate piecewise constant solutions (such as the one depicted in Figure 3b). The analysis of the classifier weights and of the eigenvalue spread of the evolved classifiers showed various phenomena underlying the evolution of accurate piecewise constant approximations for the discrete sine. First, we noted that the weights of specific classifiers tended to converge faster both because (i) in such a case of specific classifiers the spreads are generally not very large; and because, (ii) even when the spread is large, the convergence is faster for w_0 and this allows accurate approximation in the small areas of the target function covered by the specific conditions. In addition, we noted that the convergence was also faster in the subspace corresponding to positive values of w_0 and w_1, while it is slower for those weight subspaces in which w_0 and w_1 have opposite signs. Note that while a large spread in the eigenvalues highlights that in some weight subspace the convergence will be slow, it is rather difficult to characterize which weight subspace will be affected. In fact, to characterize such subspaces we need to analyze the eigenvectors of the autocorrelation matrix and how such eigenvectors map the original weight space in the space of vector $\vec{v}$.

5.4 Discussion

We have exploited some known results regarding the convergence of the modified Widrow-Hoff update to explain the results reported in the previous section. Our analysis suggests that the convergence of classifier predictions in XCSF depends on the distribution of classifier inputs and it can be characterized by means of the spread of the eigenvalues of the autocorrelation matrix R^* associated to each classifier.

In the class of problems considered so far, i.e., the approximation of a one-dimension function, assuming a uniform distribution of classifier inputs, such

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

eigenvalue spread depends both on the interval endpoints of classifier conditions and on x_0. However, the analysis highlights a more general issue: it suggests that in XCSF, as long as classifier weights are updated using the Widrow-Hoff update, the convergence speed of classifier weights is influenced by the distribution of input values. The amount of such influence can be characterized by the spread of the eigenvalues of R^* which is in turn unknown and problem dependent.

The convergence speed of the classifier prediction is a principal factor in the evolution of (accurate) maximally general classifiers in XCSF. If a classifier applies to situations for which an accurate piecewise-linear generalization exists but weight convergence is slow, it will be difficult for the classifier to emerge. In fact, due to the slow convergence of its prediction, such a classifier will tend to be evaluated as inaccurate. Accordingly, it will eventually be deleted by the evolutionary component. This is, for instance, what happened in the intervals corresponding to the slopes of the discrete sine in $[1000, 1100]$. Here XCSF was not able to evolve the most general classifier, covering the whole slope interval, because of its convergence speed. However, XCSF was still able to evolve a more specific but equally accurate solution providing a piecewise constant approximation of the slope.

As a result of our analysis, overall, we should expect that XCSF, as introduced by Wilson (2002), will perform better in those areas of the problem space where not only better generalizations exist but also when such areas are associated to weight subspaces in which convergence is fast. Note however, that even in situations of slower convergence rate, XCSF can still evolve accurate though more specific classifiers to solve the problem. In this case, the evolved solutions are still accurate but less effective than they could be, since they do not fully exploit the possibility of piecewise-linear approximation.

In the next section, we introduce three approaches to improve the convergence speed of the classifier prediction of XCSF so as to limit its dependence on the eigenvalues of the autocorrelation matrix and to favor the evolution of piecewise linear approximations.

6 Improving Generalization in XCSF

We now introduce three approaches to improve the convergence of the classifier prediction in XCSF so as to limit the influence of the input distribution on XCSF and to make generalization more effective. The first approach keeps the Widrow-Hoff update but exploits what we know about the dependence between the autocorrelation matrix R^* and the range of classifier inputs, in the case of one-dimensional functions. Specifically, it uses classifier conditions in order to preprocess input values so that both the prediction calculation and the prediction update are performed on input values that are *normalized with respect to the classifier condition*. We dub this version XCSFcn or XCSF with condition-based normalization. The other two approaches solve the issues raised by Widrow-Hoff by replacing it with better (i.e., more robust and faster converging) procedures for updating classifier weights: the *linear least squares* update and the *recursive linear least squares* update; we dub these versions XCSFls and XCSFrls respectively.

6.1 XCSF with Condition-Based Normalization

In XCSF, it is straightforward to use classifier conditions to normalize classifier inputs with respect to each specific classifier condition. While classifiers are matched against inputs in the usual problem range, both the classifier prediction computation and the update will be performed with respect to inputs between 0 and 1. In practice, the

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

structure of XCSF is kept as it is; we only modify the prediction computation and the update of classifier weights to include a preprocessing normalization step. When the prediction of a classifier *cl* needs to be computed for an input $\vec{x}$, the input $\vec{x}$ is first normalized to produce an actual input $\vec{x}'$ which is then used to compute the classifier prediction. The normalized $\vec{x}'$ is obtained from $\vec{x}$ as follows:

$$\forall i \in \{1, \ldots, |\vec{x}|\}, x'_i = \begin{cases} \frac{x_i - l_i}{u_i - l_i} & \text{when } u_i \neq l_i \\ 1 & \text{when } u_i = l_i \end{cases} \tag{18}$$

where l_i and u_i are the condition lower and the upper bound for the predicate corresponding to input i. Then the classifier prediction is computed as usual:

$$cl.p(\vec{x}) = cl.\vec{w}\vec{x}'.$$

When a classifier *cl* is updated according to an example $\langle \vec{x}, f(\vec{x}) \rangle$, the current input $\vec{x}$ is again normalized to produce $\vec{x}'$ which is then used to update the classifier weights. This approach is rather simple and it aims at preprocessing the input of the prediction update so that the overall input distribution can become more *favorable* to the Widrow-Hoff update. Note however, that this is essentially a pragmatic way to tackle the problem we discussed in the previous section. In fact, the condition-based normalization does not guarantee that in more complex problems, where there is no direct relation between the condition endpoints and the eigenvalues of R^*, the spread will not still cause slow convergence.

To test the effect of the condition-based normalization, we repeat the same experiment discussed in Section 5.3 using the condition-based normalized update. Since normalization reduces the input ranges between 0 and 1, we set x_0 to 0.5, the center of the input range. Figure 5 reports the plot of the error $\tilde{J}$ for the classifiers matching $[0, 100]$ (Figure 5a) and for the classifiers matching $[1000, 1100]$ (Figure 5b). In the two ranges, $[0, 100]$ and $[1000, 1100]$ this update reaches basically the same error values for the corresponding classifiers. Most importantly, in $[1000, 1100]$, all the classifiers converge to an overall error that is much smaller than that obtained by the usual Widrow-Hoff update (Figure 4). Finally, in Figure 6 we compare the approximation evolved for the discrete sine in $[0, 100]$ and in $[1000, 1100]$, averaged over 50 runs (Figure 6a and Figure 6b); we also report an example of single runs in Figure 6c and Figure 6d. Even in this case, the performance of XCSFcn is basically identical in the two cases.

6.2 XCSF with Linear Least Squares

Linear least squares is a well-known alternative to the Widrow-Hoff update; it is not completely new to reinforcement learning, where it has been applied to approximate TD (Bradtke and Barto 1996; Boyan 1999). Linear least squares is more efficient than Widrow-Hoff since it extracts more information from each sample and it would be expected to converge with fewer samples. Most importantly it is not sensitive to input ranges (Haykin 1986; Bradtke and Barto 1996), thus it is well suited for solving the issues discussed in the previous section.

The Widrow-Hoff update can be viewed as a gradient descent aimed at minimizing the following error function:

$$\xi(\vec{w}) = \tfrac{1}{2} e^2(t) \quad \text{where } e(t) = f(x_t) - cl.p(x_t). \tag{19}$$

Linear least squares (Haykin 1998) extends such an error function by considering the error over the previous inputs and outputs. The error function $\xi(\vec{w})$ becomes

$$\xi(\vec{w}) = \tfrac{1}{2} \textstyle\sum_{i=1}^{t} e^2(i) \quad \text{where } e(i) = f(x_i) - cl.p(x_i). \tag{20}$$

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

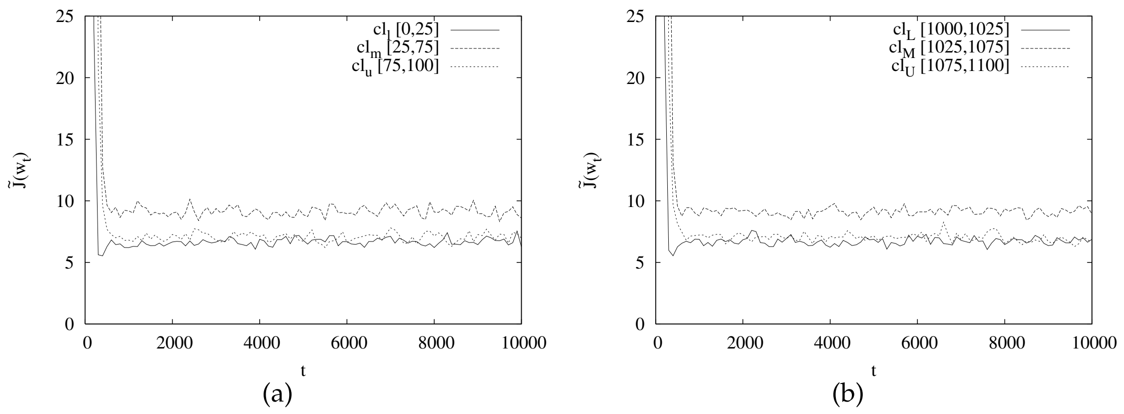


Figure 5: Widrow-Hoff with condition-based normalization: convergence of the classifier prediction with condition-based normalization for (a) classifiers matching $I_1 = [0, 100]$ and for (b) classifiers matching $I_2 = [1000, 1100]$. Plots are averages over 50 runs.

Let $X_t = [\vec{x_1}, \vec{x_2}, \ldots, \vec{x_t}]^T$ be vector of inputs until time t, and let $\vec{f}(t) = [f(\vec{x_1}), f(\vec{x_2}), \ldots, f(\vec{x_t})]$ be the vector of all the desired outputs. The weight vector $\vec{w}$ that minimizes the error function in Equation 20, is computed as the solution of the following equation:

$$(X_t^T X_t)\vec{w} = X_t^T \vec{f}(t). \tag{21}$$

The weight vector $\vec{w}$ can be either determined by computing the *pseudoinverse* X^+ of X ($X^+ = (X_t^T X_t)^{-1} X_t^T$), or by applying the Singular Value Decomposition to $(X_t^T X_t)$ (Press, Teukolsky, Vetterling, and Flannery 2002). The least squares update requires that all the inputs used for updating are stored. But in general we are interested in an incremental update; for this purpose least squares is implemented by keeping a vector X_t^n of the latest n visited inputs and a vector f_t^n of the corresponding n desired outputs (Haykin 1998). The incremental version of least squares is performed as follows. Given

$$\begin{aligned} X_t^n &= [\vec{x}_{t-n+1}, \vec{x}_{t-n+2}, \ldots, \vec{x_t}]^T, \\ Y_t^n &= [f(\vec{x}_{t-n+1}), f(\vec{x}_{t-n+2}), \ldots, f(\vec{x_t})]^T, \end{aligned}$$

the weight vector $\vec{w}$ to update the current classifier weight vector is computed as the solution of the following equation:

$$(X_t^n)^T X_t^n \vec{w} = (X_t^n)^T Y_t^n.$$

Then, the current weight vector is updated as

$$\vec{w_t} = (1 - \eta)\vec{w}_{t-1} + \eta\vec{w}.$$

This formulation of the least squares allows us to use a small window, given an adequately small value of the learning rate η.

To add linear least squares to XCSF in each classifier we need to keep trace of the last n encountered inputs and of the corresponding n outputs. Accordingly, classifiers

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

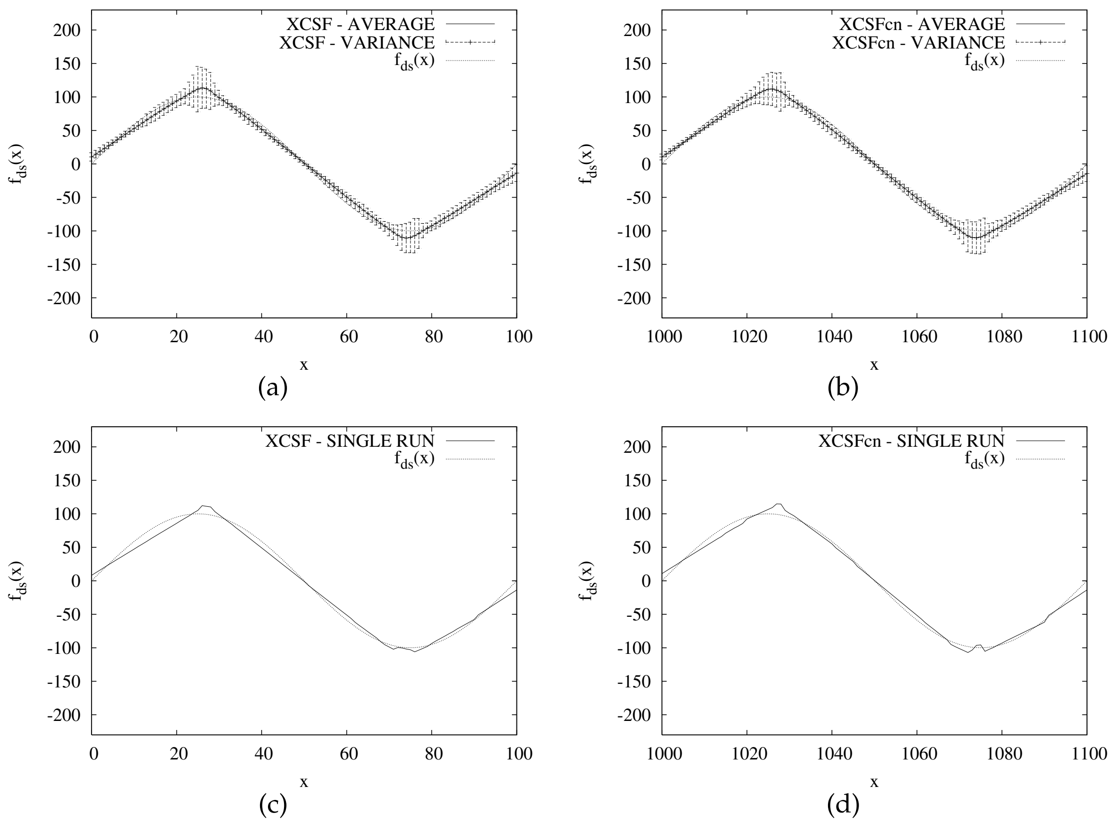


Figure 6: XCSFcn applied to the discrete sine with a population size $N = 800$ and an error threshold $\epsilon_0 = 10$. Comparison of XCSFcn approximation (solid line) against the target function (dashed line) when $x \in [0, 100]$, (a) reports the average over 50 runs, (c) reports a typical run; when $x \in [1000, 1100]$, (b) reports the average over 50 runs, (d) reports a typical run.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

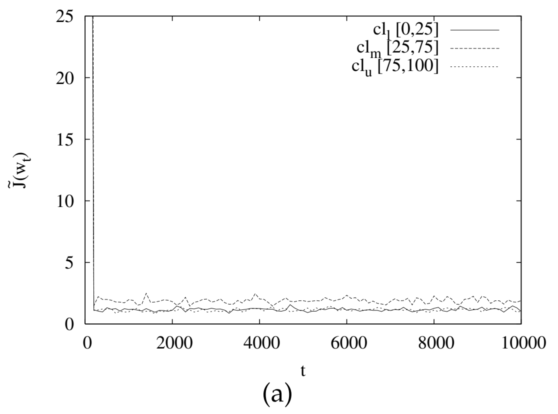
Algorithm 3 XCSFls: Update classifier *cl* with linear least squares

```
1: procedure UPDATE_PREDICTION($cl$, $s$, $P$)
2:     add($cl.X$,$s$);                              ▷ Add the current input $s$ to $X_t$
3:     add($cl.Y$,$P$);                              ▷ Add the current target $P$ to $Y_t$
4:     $A \leftarrow cl.X^t \times cl.X$;
5:     $\vec{b} \leftarrow cl.X^t \times cl.Y$;
6:     $\vec{w} \leftarrow$ SOLVE$(A, \vec{b})$;               ▷ Compute the new weight vector
7:     for $i \in \{0, \ldots, |s|\}$ do              ▷ Update the classifiers weights
8:         $cl.w_i \leftarrow (1 - \eta)cl.w_i + \eta w_i$;
9:     end for
10: end procedure
```

are enriched with two new parameters, the vector X of the last n inputs, and the vector Y storing the n outputs of the elements in X; both X and Y are FIFO queues of n elements, implemented as circular arrays. The update of the classifier prediction with linear least squares is reported as Algorithm 3. The procedure SOLVE encapsulates the algorithm for solving the linear system which in our implementation is based on the Singular Value Decomposition (SVD) (Press, Teukolsky, Vetterling, and Flannery 2002) as implemented in the GSL/BLAS package (Galassi et al. 2004).

Also for XCSFls we repeat the experiments performed in Section 5.3 for the Widrow-Hoff update and in the previous section for XCSFcn. First, we analyze the convergence of the linear least squares update on the three classifiers cl_l, cl_m, and cl_u applying in $[0, 100]$ and on the three classifiers cl_L, cl_M, and cl_U, applying in $[1000, 1100]$ (see Section 5.3). For all the experiments from here on, the size of X and Y is set to 50 examples. We remind the reader that in this case we only perform prediction updates on these six classifiers, we do not actually use XCSFls. Figure 7 compares the error $\tilde{J}$ of classifiers applying in $[0, 100]$ (Figure 7a) and the error $\tilde{J}$ of those applying in $[1000, 1100]$ (Figure 7b). As can be noted, the error plots for the two intervals of classifiers are basically the same, i.e., the classifiers in the two intervals behave the same; as in the experiments with the Widrow-Hoff and XCSFcn the classifiers cl_m and cl_M corresponding to the central intervals converge to a higher error than the other classifiers. Most importantly we note that with the linear least squares update *all* the classifiers rapidly converge to a much lower error than that obtained by Widrow-Hoff update (Figure 4) and by the condition-based normalization (Figure 5). Figure 8 compares the approximation evolved by XCSFls for the discrete sine in $[0, 100]$ (Figure 8a) and in $[1000, 1100]$ (Figure 8b). In the two intervals, XCSFls performance is basically the same. When we compare the $\overline{MAE}$ and $\overline{MSE}$ in I_1 and I_2 the two cases, we find that both the parametric tests (t-test and F-test) and the non-parametric tests (Wilcoxon's and Mood's tests) return no significant difference with a confidence level of the 99.999% (Glantz and Slinker 2001). Figure 9 reports the distributions of the weights w_1, responsible for the classifiers' approximation slope, in the two intervals $[0, 100]$ (upper plot) and $[1000, 1100]$ (lower plot). Clearly the two distributions are almost identical. As in the case of XCSFcn the vast majority of classifier weights are distributed around two values, corresponding to the slopes that best approximate the three sections of the sine curve. More classifiers have a weight around 4.1, that represents the best slope to approximate two of the three intervals that correspond to the main generalization sections of the sine curve; (those at the beginning and the end of the input range).

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021



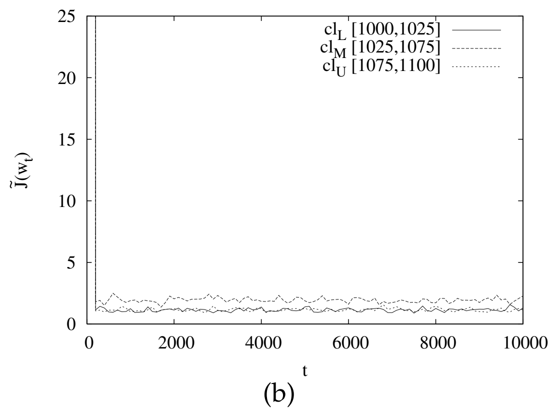


Figure 7: Linear least squares: convergence of the classifier prediction for (a) classifiers matching $I_1 = [0, 100]$ and for (b) classifiers matching $I_2 = [1000, 1100]$. Plots are averages over 50 runs.

6.3 XCSF with Recursive Least Squares

Linear least squares requires a trace of the past n inputs (the vector X_T^n) and of the corresponding n outputs (in the vector Y_T^n) that are used to update the weights. Recursive linear least squares (Haykin 1986) implements linear least squares through recursive computation, without using the vectors X_T^n and Y_T^n. Recursive linear least squares keeps an estimate V of the matrix $(X_t^T X_t)^{-1}$, needed to update the weight vector, that is computed recursively. At time step t, given the current input x_t and the target value y_t, recursive least squares updates the weight vector $\vec{w}$ as

$$\vec{w}_t = \vec{w}_{t-1} + \vec{k}_t[y_t - \vec{x}_t \vec{w}_{t-1}],$$

where, $\vec{k}_t$ is the *gain vector* computed as

$$\vec{k}_t = \frac{V_{t-1}\vec{x}_t}{1 + \vec{x}_t^T V_{t-1} \vec{x}_t}, \tag{22}$$

while matrix V_t is computed recursively by

$$\vec{V}_t = \left[I - \vec{k}_t x_t^T\right] V_{t-1}. \tag{23}$$

Recursive least squares implements least squares rigorously and it can obtain the same results when the matrix V and the weight vector $\vec{w}$ are initialized as follows:

$$V_0 = \left(\sum_i \vec{x}_i^T \times \vec{x}_i\right)^{-1}, \tag{24}$$

$$\vec{w}_0^T = V_0 \sum_i \vec{x}_i^T \times y_i. \tag{25}$$

Alternatively, we can avoid the computation of the inverse in Equation 24, by setting (Haykin 1998)

$$V_0 = \delta_{rls} \times I, \tag{26}$$

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

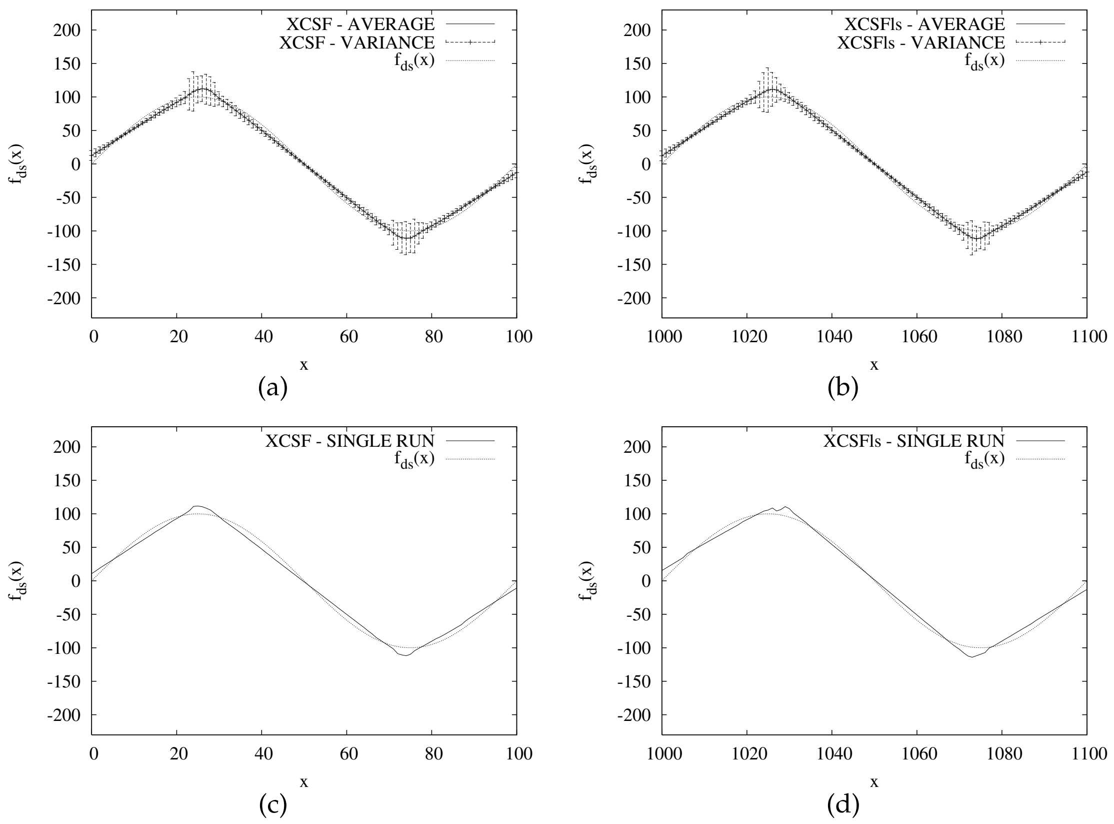


Figure 8: XCSFls applied to the discrete sine with a population size $N = 800$ and an error threshold $\epsilon_0 = 10$. Comparison of XCSFls approximation (solid line) against the target function (dashed line) when $x \in [0, 100]$, (a) reports the average over 50 runs, (c) reports a typical run; when $x \in [1000, 1100]$, (b) reports the average over 50 runs, (d) reports a typical run.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

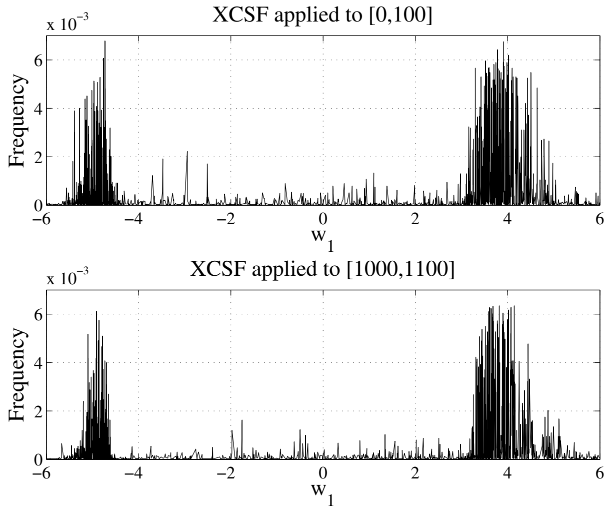


Figure 9: XCSFls applied to the discrete sine. Distribution of weight w_1 for I_1 (upper plot) and for I_2 (lower plot).

where I is the identity matrix of dimension $|x| \times |x|$ and $\delta_{rls} > 0$. An higher δ_{rls}, denotes that initial parameterization is uncertain, accordingly, initially the algorithm will use a higher, thus faster, update rate ($\vec{k}_t$). A lower δ_{rls}, denotes that initial parameterization is rather certain, accordingly the algorithm will use a slower update. It is interesting to note that recursive least squares is a special case of Kalman filtering to the case of a fixed target state and no control signal applied to the state variables (the weight vector $\vec{w}$) as in the tasks considered here (Goodwin and Sin 1984).

We now extend XCSF with recursive least squares by adding to each classifier a matrix V (of size $|x| \times |x|$) that replaces the vectors X_T^n and Y_t^n in XCSFls classifiers. The update of XCSF with recursive least squares, named XCSFrls, is reported as Algorithm 4.

Algorithm 4 XCSF: update procedure with RLS

```
1:  procedure UPDATE_PREDICTION($cl$, $s$, $P$)
2:      error ← $P - cl.p(s)$;                                   ▷ Compute the current error
3:      $x(0) \leftarrow x_0$;                                    ▷ Build $\vec{x}$ by adding $x_0$ to $s$
4:      for $i \in \{1, \ldots, |s|\}$ do
5:          $x(i) \leftarrow s(i)$;
6:      end for
7:      $\beta_{rls} \leftarrow 1 + \vec{x} \times cl.V \times \vec{x}^T$;    ▷ Compute the matrix update rate $\beta_{rls}$
8:      $cl.V \leftarrow cl.V - \beta_{rls}^{-1} \times cl.V \times \vec{x}^T \times \vec{x} \times cl.V$;   ▷ Update $cl.V$
9:      $\vec{k}^T \leftarrow cl.V \times \vec{x}^T$;
10:     for $i \in \{0, \ldots, |s|\}$ do                        ▷ Update classifier's weights
11:         $cl.w_i \leftarrow cl.w_i + \vec{k}(i) \times error$;
12:     end for
13: end procedure
```

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

It is worth comparing the complexity of the linear least squares and the recursive least squares both in terms of memory required for each classifier and time required by each classifier update. For each classifier, linear least squares stores the last n classifier inputs and the corresponding n outputs (the matrix X_t^n and the vector Y_t^n), thus its complexity in terms of memory is $O(n \times |\vec{x}|)$, where $|\vec{x}|$ is the size of the input vector; recursive least squares stores only the matrix V_t which is $|\vec{x}| \times |\vec{x}|$, thus its complexity is $O(|x|^2)$. We can argue that the spatial complexity of recursive least squares is lower than that of linear least squares approach, since the number n of observations needed by linear least squares is generally larger than the number of inputs $|\vec{x}|$ (Haykin 1986). With respect to the time required for each update, the most time expensive operation required by linear least squares is the matrix inversion (Algorithm 3, line 6) which is $O(|x|^3)$; instead recursive least squares requires only matrix multiplication which is $O(|x|^2)$. Therefore, recursive least squares is less complex than linear least squares both in terms of memory and time requirements. In the next section, we compare these two methods, together with the condition-based normalization, also in terms of approximation accuracy provided.

6.4 Additional Experiments

We compare the four versions of XCSF on the following functions:

$$f_{s3}(x) = 100\left(\sin(\frac{2\pi x}{100}) + \sin(\frac{4\pi x}{100}) + \sin(\frac{6\pi x}{100})\right), \tag{27}$$

$$f_{s4}(x) = 100\left(\sin(\frac{2\pi x}{100}) + \sin(\frac{4\pi x}{100}) + \sin(\frac{6\pi x}{100}) + \sin(\frac{8\pi x}{100})\right), \tag{28}$$

$$f_{abs}(x) = 100\left|\sin(\frac{2\pi x}{100}) + \left|\cos(\frac{2\pi x}{100})\right|\right|. \tag{29}$$

Equation 27 and Equation 28 are Koza's Sinus Three and Sinus Four (1992) adapted for integers as done by Wilson (2002) for the sine; Equation 29 is taken from (Zelinka 2002) and adapted for integers. The main parameter settings for all the experiments are summarized in Table 2 where the average mean absolute error ($\overline{MAE}$) is also reported; for XCSFls the size of the data windows is set to 50; for XCSFrls, δ_{rls} is set to 100; all the parameters not included in Table 2 have been set as in the previous experiments. Note that, to better highlight the differences among the four versions of XCSF, in the experiments discussed here we use a population size that is half that used by Wilson (2001a) for the discrete sine.

In $f_{s3}(x)$, when the error threshold ϵ_0 is 10, all the three proposed systems reach accurate approximations with average errors below the target threshold (Table 2). When the error threshold ϵ_0 is lowered to 5, XCSF evolves approximations with an average error higher than ϵ_0; while XCSFcn and XCSFls evolve approximations that on the average are accurate in that their average error is smaller than ϵ_0. To obtain an $\overline{MAE}$ below ϵ_0=5, in $f_{s3}(x)$ XCSF needs a population size N of 800 classifiers.

Figure 10 reports the approximation of $f_{s3}(x)$ for $\epsilon_0 = 10$ evolved by XCSF (Figure 10a) with that evolved by XCSFls (Figure 10b). The solutions evolved by the two versions are radically different as can be noted by considering the comparison between the best and worst solutions evolved by XCSF (Figure 10c) and XCSFls (Figure 10d). Again, with the Widrow-Hoff update solutions generally consist of classifiers either providing piecewise constant approximations made of flat segments, or of overly spe-

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

$f(x)$	I	ϵ_0	N	XCSF	XCSFcn	XCSFls	XCSFrls
$f_{s3}(x)$	$[950, 1050]$	10	400	8.6 ± 1.1	5.5 ± 0.6	6.0 ± 0.3	6.1 ± 0.3
$f_{s3}(x)$	$[950, 1050]$	5	400	8.0 ± 1.1	3.2 ± 0.7	3.2 ± 0.2	3.2 ± 0.3
$f_{s4}(x)$	$[950, 1050]$	10	400	11.2 ± 1.6	11.3 ± 1.9	6.6 ± 0.4	6.1 ± 0.4
$f_{s4}(x)$	$[950, 1050]$	5	400	11.7 ± 1.7	11.7 ± 2.5	3.6 ± 1.4	3.4 ± 1.2
$f_{abs}(x)$	$[950, 1050]$	5	400	4.1 ± 0.5	4.1 ± 0.4	2.4 ± 0.2	2.5 ± 0.2

Table 2: Mean absolute error for XCSF, XCSFcn XCSFls, and XCSFrls: $f(x)$ is the target function; I is the function domain; ϵ_0 is the error threshold reported as a percentage of the function range; for each system (XCSF, XCSFcn, XCSFls, and XCSFrls) we report the value of the average mean absolute error with the standard deviation ($\overline{MAE} \pm \sigma$). Statistics are averages over 50 runs.

cific classifiers applying in one point. As an example, in Figure 11a we report the population that provides the best evolved approximation, reported in Figure 10c.[2] In contrast XCSFls evolves classifiers representing piecewise linear approximations covering large parts of the function domain as can be noted from the population depicted in Figure 11b, which represents the best approximation in Figure 10d. From our experiments, XCSFls also appears to be more stable in that the difference between the best and the worst approximation is much less than that evolved by XCSF. In $f_{s3}(x)$, XCSFcn performs basically like XCSFls (not shown).

As the problem becomes more complex, moving to $f_{s3}(x)$ to $f_{s4}(x)$, the ability of the system to evolve proper generalizations becomes more important. Thus, it becomes more important for the system to convergence quickly to those weight configurations that allow higher generalization. As a consequence, in $f_{s4}(x)$ the models based on the Widrow-Hoff update (XCSF and XCSFcn) perform worse than XCSFls and XCSFrls both for $\epsilon_0 = 10$ and $\epsilon_0 = 5$, obtaining an average error higher than the threshold ϵ_0, very similar to that of XCSF (see Table 2). In contrast, XCSFls and XCSFrls always perform better than XCSF and XCSFcn. Figure 12 reports the approximation evolved by XCSF (Figure 12a) with that evolved by XCSFls (Figure 12b) when ϵ_0 is 10. The solutions evolved by the two models are again radically different, as can be best noted by considering the comparison between the best and worst solutions evolved by XCSF (Figure 10c) and XCSFls (Figure 10d).[3]

In $f_{abs}(x)$, all three models reach an average absolute error below the threshold ϵ_0 (Table 2). Figure 13 reports the approximation evolved by XCSF (Figure 13a) with that evolved by XCSFls (Figure 13b); Figure 13c and Figure 13d report the best and worst approximations evolved by XCSF and XCSFls respectively. Also in this case, the solutions evolved by XCSF and XCSFls are qualitatively different, and the overall behavior is the same observed in the previous experiments: XCSF evolves piecewise constant approximations of $f_{abs}(x)$, while XCSFls evolves classifiers representing piecewise linear approximations.

For all the performed experiments, in Table 3 we report (i) the average number of (macro) classifiers in the final populations (column $|[P]|$) with the corresponding

[2]Note that the solution reported in Figure 11a does not take into account either classifier fitness or classifier numerosity which are used to compute the XCS system prediction; accordingly, it is rather difficult to map the population depicted in Figure 11a to its approximation in Figure 10c. For instance, the classifier in Figure 11a that covers the section between 980 and 1000 has both a small numerosity and a small fitness; thus its contribution on the overall prediction is almost null.

[3]Note that, although XCSF reaches an average mean absolute error higher than the error threshold ϵ_0, the best prediction depicted in Figure 10c is actually accurate, having a mean absolute error of 8.5.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

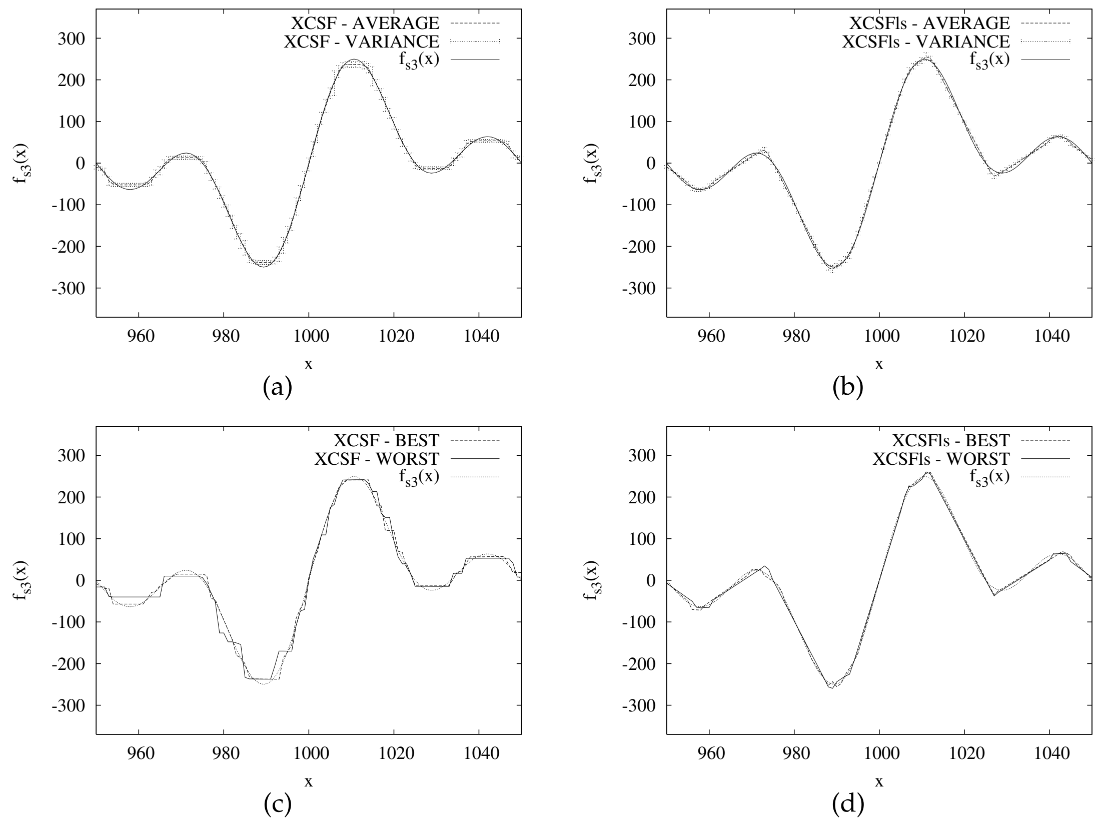


Figure 10: Comparison of XCSF and XCSFls applied to $f_{s3}(x)$ with $N = 400$ and $\epsilon_0 = 10$: (a) XCSF approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (b) XCSFls approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (c) best (dashed line) and worst (solid line) approximations evolved by XCSF; (d) best (dashed line) and worst (solid line) approximations evolved by XCSFls. Curves in (a) and (b) are averages over 50 runs.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

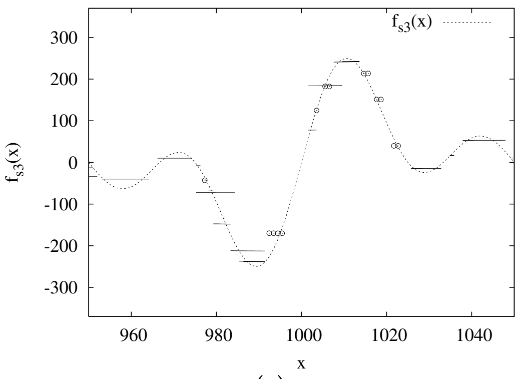
		XCSF	XCSFcn	XCSFls	XCSFrls
$f(x)$	ϵ_0	$\|[P]\| \pm \sigma$	$\|[P]\| \pm \sigma$	$\|[P]\| \pm \sigma$	$\|[P]\| \pm \sigma$
$f_{s3}(x)$	10	47.2 ± 3.4	19.4 ± 3.2	21.8 ± 3.8	22.3 ± 3.6
$f_{s3}(x)$	5	55.8 ± 5.0	24.5 ± 2.9	24.5 ± 3.5	24.9 ± 2.7
$f_{s4}(x)$	10	49.1 ± 4.3	49.5 ± 4.4	24.0 ± 3.3	24.0 ± 3.2
$f_{s4}(x)$	5	48.9 ± 4.3	57.0 ± 4.1	29.7 ± 4.2	29.6 ± 3.3
$f_{abs}(x)$	5	45.1 ± 4.2	45.3 ± 4.0	22.1 ± 3.6	20.9 ± 3.1

(a)

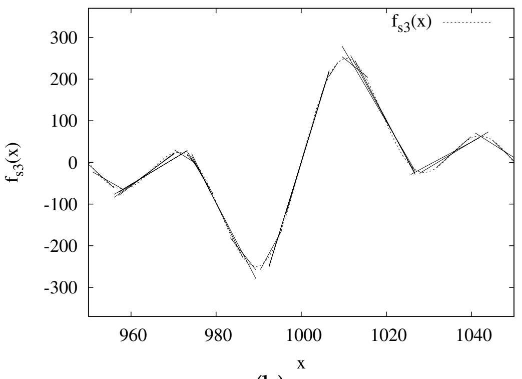
		XCSF	XCSFcn	XCSFls	XCSFrls
$f(x)$	ϵ_0	G([P])$\pm\sigma$	G([P])$\pm\sigma$	G([P])$\pm\sigma$	G([P])$\pm\sigma$
$f_{s3}(x)$	10	4.1 ± 3.5	10.6 ± 3.9	12.6 ± 4.3	12.7 ± 4.3
$f_{s3}(x)$	5	3.3 ± 2.9	7.5 ± 3.2	8.5 ± 3.5	8.6 ± 3.6
$f_{s4}(x)$	10	3.9 ± 3.1	3.9 ± 3.2	9.4 ± 3.4	9.5 ± 3.4
$f_{s4}(x)$	5	4.0 ± 3.1	3.2 ± 2.8	6.5 ± 3.0	6.5 ± 3.0
$f_{abs}(x)$	5	4.5 ± 4.1	4.6 ± 4.1	15.0 ± 2.5	15.1 ± 2.5

(b)

Table 3: Generalization with XCSF, XCSFcn, XCSFls, and XCSFrls: $f(x)$ is the target function; ϵ_0 is the error threshold reported as a percentage of the function range; (a) $|[P]| \pm \sigma$ is the average size of the evolved solution with the corresponding standard deviation; (b) $G([P]) \pm \sigma$ is the average generality of classifiers in the evolved solution with the corresponding standard deviation. Population size N is 400 classifiers; functions are defined over the interval $[950, 1050]$. Statistics are averages over 50 runs.



(a)



(b)

Figure 11: XCSF and XCSFls applied to $f_{s3}(x)$: (a) best population evolved by XCSF and (b) best population evolved by XCSFls. Segments identify the approximations provided by classifiers; circles represent classifiers matching one point. The corresponding approximations are reported in Figure 10c and Figure 10d respecitvely.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

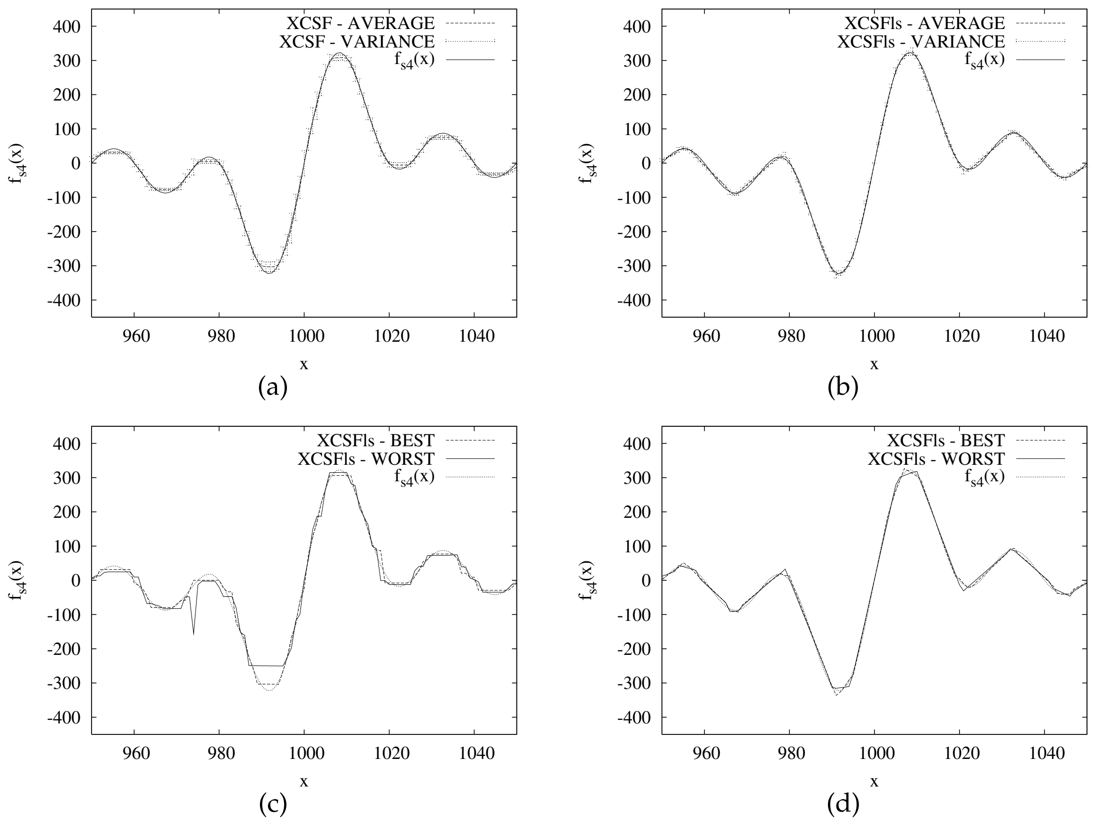


Figure 12: Comparison of XCSF and XCSFls applied to $f_{s4}(x)$ with $N = 400$ and $\epsilon_0 = 10$: (a) XCSF approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (b) XCSFls approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (c) best (dashed line) and worst (solid line) approximations evolved by XCSF; (d) best (dashed line) and worst (solid line) approximations evolved by XCSFls. Curves in (a) and (b) are averages over 50 runs.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

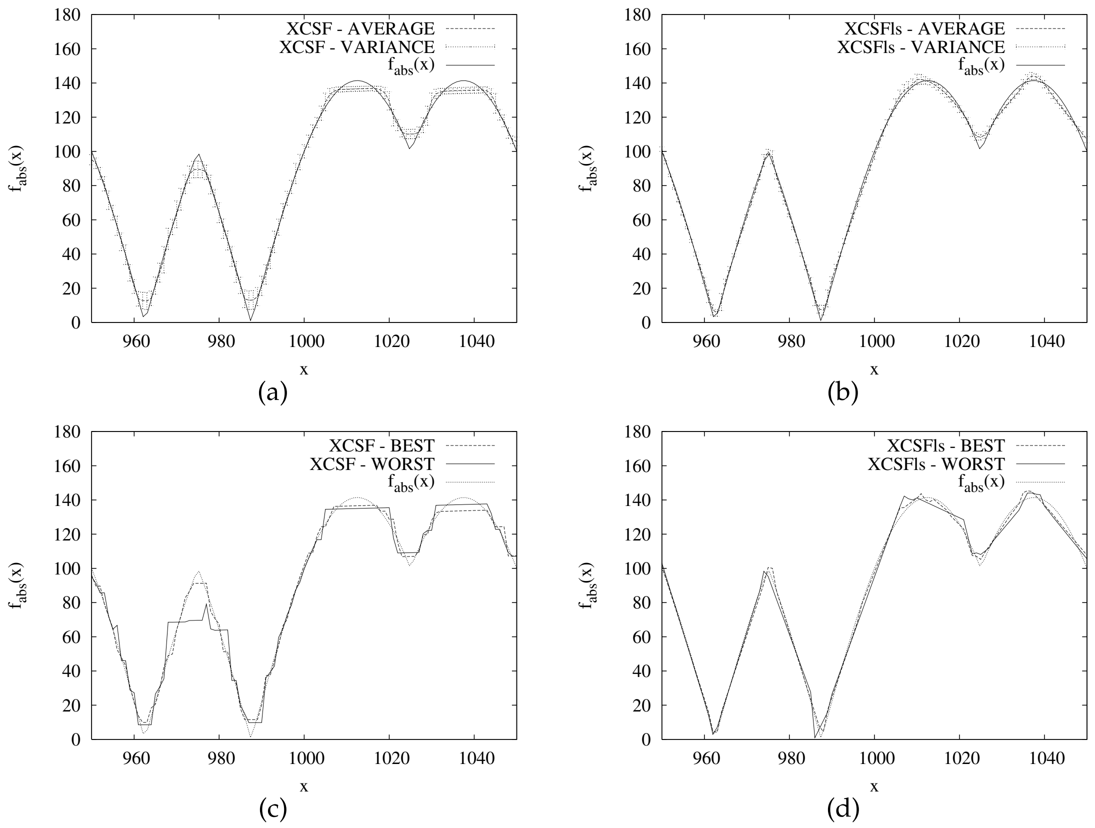


Figure 13: Comparison of XCSF and XCSFls applied to $f_{abs}(x)$ with $N = 400$ and $\epsilon_0 = 5$: (a) XCSF approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (b) XCSFls approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (c) best (dashed line) and worst (solid line) approximations evolved by XCSF; (d) best (dashed line) and worst (solid line) approximations evolved by XCSFls. Curves in (a) and (b) are averages over 50 runs.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

standard deviation ($\pm\sigma$); (ii) the average generality of the classifiers in the populations evolved in all the experiments (column $G([P])$) with the corresponding standard deviation ($\pm\sigma$). The value of $G([P])$ is computed as the average number of inputs matched by all the classifiers evolved in the 50 experiments, therefore the values of $G([P])$ are basically averages over $N \times 50$ values. As can be noticed, XCSFls and XCSFrls generally evolve solutions that are more compact (the values of $|[P]|$ are generally smaller) since they consist of more general classifiers (the values of $G([P])$ are generally larger).

Statistical Analysis. We apply a *one-way* analysis of variance or ANOVA (Glantz and Slinker 2001) to test whether the differences in approximation accuracy and generalization capabilities observed for the four versions of XCSF (Table 2) are statistically significant. Then, we apply a series of (typical) post-hoc procedures (namely Tukey, Scheffé, and Bonferroni) to analyze the differences among different versions of XCSF. Note that we use ANOVA instead of plain t-test since we are interested in checking whether there is an overall difference in the performance of the four versions. The t-test would test just whether two versions perform statistically differently in one specific setting (e.g., $f_{s3}(x)$ with ϵ_0=5). First we analyze the mean absolute error ($\overline{MAE}$). The ANOVA test performed on the data of mean absolute error shows that some versions of XCSF perform significantly differently, with a confidence level of the 99.99%. All the three subsequent *post-hoc* procedures (Scheffé, LSD, and Tukey) show that the performance of XCSFls and XCSFrls is not significantly different, that is, in term of absolute error the linear least squares and the recursive least squares perform the same. The same post-hoc procedures show also that the mean absolute error obtained by XCSF is significantly different from all the other three versions; the same holds for XCSFcn. When we apply the same analysis to the data of the population size (column $|[P]|$ in Table 2) and to the data of the population generality (column $G([P])$ in Table 2) we obtain similar results: (i) the difference observed for all the four versions of XCSF is statically significant; (ii) the difference between XCSFls and XCSFrls both in terms of size of the evolved solutions both in terms of generality of the classifiers in the final populations is not statistically significant, that is, the two versions are basically equivalent; (iii) the difference between least squares approaches (XCSFls and XCSFrls) and Widrow-Hoff approaches (XCSF and XCSFcn) is statistically significant, that is, least squares approaches significantly improve the generalization capability of XCSF.

6.5 Discussion

The three versions of XCSF we introduced represent different approaches to the issue raised in XCSF by the mathematical properties of Widrow-Hoff update (Widrow and Hoff 1988). The first approach, condition-based normalization, was introduced as a rather simple approach to keep the Widrow-Hoff update in XCSF while limiting, in the case of one-dimensional functions, the influence of the input range over the speed of weight convergence. The other two approaches, linear least squares and recursive linear least squares, solve the issues raised by the Widrow-Hoff update by replacing it with more robust and faster-converging procedures for the update of classifier weights. It is not the goal of this section, nor of this paper, to state whether one of these approaches is the best replacement of the usual Widrow-Hoff update in XCSF; an absolute comparison of the three approaches is infeasible. Nevertheless, we can summarize the pros and the cons of these methods.

On the one hand, condition based normalization necessitates minimal modifications to the original structure; it does not require additional memory, and it appears to perform better than Widrow-Hoff in some of the simple problems we considered here.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

On the other hand, this approach is still based on Widrow-Hoff and therefore it does not guarantee that in more complex problems the spread of eigenvalues will stay large so that the classifier predictions will converge slowly. In addition, we note that a normalization performed only on the inputs, but not on the outputs, drastically modifies the weight space.

In contrast, the linear least squares and the recursive least squares updates have a strong theoretical base, and they have been widely studied and applied, including to Reinforcement Learning (Bradtke and Barto 1996); Most importantly, they are more robust with respect to the distribution of classifier inputs and have higher convergence speed, which in terms of XCSF means more accurate information to the genetic algorithm. However, they require more computational resources both in terms of (i) memory, since classifiers have to maintain extra information about past inputs; and in terms of (ii) time, since for each update different matrix operations are required. However, the complexity of recursive least squares is lower than that of the plain linear least squares.

Thus, we suggest that in applications without strict bounds on computational resources, both in time and in memory, least squares updates are certainly the best choice since they are more theoretically grounded and are more effective in feeding the genetic algorithm with reliable information.

7 XCSF Beyond Linear Approximation

So far, we have applied XCSF to evolve piecewise linear approximations of the prediction function. The same approach can be easily extended to allow polynomial approximations. Let us consider the problem we tackled so far, i.e., the approximation of a monodimensional function $f(x)$. Given the current input x, in XCSF the classifier prediction is computed as

$$cl.p(\vec{x}) = w_0x_0 + w_1x,$$

where $\vec{x} = \langle x_0, x\rangle$, and x_0 is the usual fixed constant. We can introduce a quadratic term in the approximation evolved by XCSF so that

$$cl.p(\vec{x}) = w_0x_0 + w_1x + w_2x^2. \tag{30}$$

To learn the new set of weights we use the usual XCSF update procedure (e.g., Widrow-Hoff) applied to the input vector $\vec{y}$ obtained by preprocessing the actual input $\vec{x}$, as $\vec{y} = \langle x_0, x, x^2\rangle$.

The same approach can be generalized to allow the approximation of any polynomials. Given a function $f(x)$ of one variable, to use XCSF to evolve an approximation $\hat{f}(x)$ based on a polynomial of order k, we define

$$\vec{y} = \langle x_0, x, x^2, \ldots, x^k\rangle$$

and apply XCSF to the newly defined input space. When more variables are involved we have two possibilities. On the one hand, we can use XCSF to evolve a solution built as a sum of different polynomials, one for each variable. In this case, given n input variables $(x_1, \ldots, x_n)$, so that $\vec{x} = \langle x_0, x_1, \ldots, x_n\rangle$, we define

$$\vec{y} = \langle x_0, x_1, x_1^2, \ldots, x_1^k, \ldots, x_n, x_n^2, \ldots, x_n^k\rangle.$$

On the other hand, we can use the *multivariate polynomial* in n variables and define the new input $\vec{y}$ accordingly. In this case, with two input variables, we would have

$$\vec{y} = \langle x_0, x_1, x_2, x_1x_2, \ldots, x_1^kx_2^k\rangle.$$

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

$f(x)$	ϵ_0	XCSFls			XCSFrls		
		$\overline{MAE} \pm \sigma$	$\lvert[P]\rvert \pm \sigma$	G([P])$\pm\sigma$	$\overline{MAE} \pm \sigma$	$\lvert[P]\rvert \pm \sigma$	G([P])$\pm\sigma$
$f_{s3}(x)$	10	5.1 ± 0.5	19.9 ± 4.0	21.6 ± 3.0	5.2 ± 0.6	18.9 ± 3.7	15.7 ± 4.8
$f_{s3}(x)$	5	4.0 ± 0.8	18.3 ± 3.9	18.4 ± 2.6	2.9 ± 0.3	23.4 ± 3.5	9.4 ± 3.7
$f_{s4}(x)$	10	5.8 ± 0.6	21.7 ± 4.3	16.4 ± 2.5	6.0 ± 0.4	21.5 ± 3.9	10.4 ± 3.9
$f_{s4}(x)$	5	4.8 ± 1.7	17.9 ± 3.8	13.6 ± 2.4	3.4 ± 0.8	28.6 ± 3.8	6.8 ± 3.0
$f_{abs}(x)$	5	1.9 ± 0.4	17.1 ± 3.6	22.5 ± 10.2	1.9 ± 0.3	11.6 ± 2.9	22.3 ± 10.4

Table 4: Summary for the experiments conducted with XCSFls and XCSFrls with quadratic approximations: $f(x)$ is the target function; I is the function domain; ϵ_0 is the error threshold reported as a percentage of the function range; $\overline{MSE} \pm \sigma$ is the average mean square error with the standard deviation; $|[P]| \pm \sigma$ is the average size of the evolved solution with the corresponding standard deviation; $G([P]) \pm \sigma$ is the average generality of classifiers in [P] with the corresponding standard deviation. All the functions are defined over the interval $[950, 1050]$; all the experiments use a population size N of 400 classifiers. Statistics are averages over 50 runs.

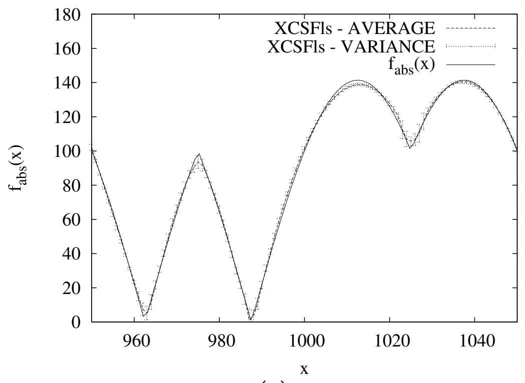
When using this approach we obtain approximations in which mixed terms, involving multiplication of different variables, appear. Nevertheless, a major drawback of this approach is that the size of $\vec{y}$ dramatically grows as the number of actual variables in $\vec{x}$ increases.

We test XCSF with quadratic approximation and we apply XCSFls and XCSFrls to the three functions discussed in the previous section, $f_{s3}(x)$, $f_{s4}(x)$, and $f_{abs}(x)$, with the same parameters used in the previous experiments.

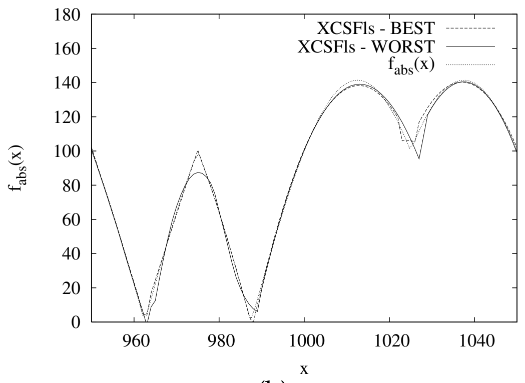
Table 4 reports the statistics for all the experiments. As can be noted, the accuracy of the solutions evolved by XCSFls and XCSFrls is similar (column $\overline{MAE}$ in Table 4). However, it seems that XCSFls evolves more general solutions: the number of classifiers in the final populations of XCSFls is smaller in $f_{s3}(x)$ and $f_{s4}(x)$ when ϵ_0=5 (column $|[P]|$ in Table 4), while the average classifier generality is usually higher (column $G([P])$ in Table 4). We apply the same statistical procedure used in Section 6 to test whether the differences between XCSFls and XCSFrls reported in Table 4 are statistically significant. In this case, the analysis of variance shows that XCSFrls evolves solutions that are more accurate than XCSFls and the difference is statistically significant with a confidence level of the 99.99% ($F = 9.420$) (Glantz and Slinker 2001). On the other hand, XCSFls evolves populations that are sometimes smaller (column $|[P]|$ in Table 4) than those evolved by XCSFrls, and these differences are statically significant according to a confidence level of the 99.99% ($F = 13.910$). Finally, XCSFls evolves classifiers with a higher generality than those evolved by XCSFrls (column $G([P])$ in Table 4) and the difference is again statistically significant with a confidence level of 99.99% ($F = 31004.2$).

As an example, Figure 14 and Figure 15 report the average approximation evolved for $f_{abs}(x)$ (Figure 14a), the most accurate and the least accurate approximations produced (Figure 14b), the population corresponding to the most accurate approximation (Figure 15a), and that corresponding to the least accurate approximation (Figure 15b). As can be noted, the most accurate solutions tend to produce classifiers which do not exploit the quadratic term where not needed (e.g., Figure 14a). However, since the error threshold is reasonably high, solutions with classifiers exploiting the quadratic can still be accurate and therefore also evolved. When the error threshold is decreased, e.g., with $\epsilon_0 = 2$, then the quadratic terms tend to zero in the linear section (not showed

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021



(a)



(b)

Figure 14: XCSFls with quadratic approximation applied to $f_{abs}(x)$ with $N = 400$ and $\epsilon_0 = 5$: (a) XCSFls approximation (dashed line) with variance (light dashed bars) against the target function (solid line); (b) best (dashed line) and worst (solid line) solutions evolved by XCSFls. The curve in (a) is an average over 50 runs.

here). Again, we wish to point out that the plots of the populations in Figure 15 do not take into account either classifier fitness or classifier numerosity; thus it is difficult to map the populations (Figure 15) to the approximations they represent (Figure 14b). For example, in Figure 15a there is a classifier that represents a quadratic approximation in the linear part of the target function, but that classifier has only a slight influence on the overall approximation since it has a small fitness and a small numerosity.

8 Conclusions

In this paper, we developed a better understanding of generalization in XCSF and introduced three ways to improve it. We presented simple experiments showing that while XCSF can evolve accurate approximations, the associated generalizations are not as efficient and reliable as might be expected. We analyzed these results from a theoretical standpoint showing a weakness in the original classifier update (Wilson 2001a; Wilson 2002). Our analysis suggested that the convergence of classifier predictions in XCSF depends on the distribution of classifier inputs and can be characterized by means of the spread of the eigenvalues of the autocorrelation matrix associated with each classifier. When the spread is large the convergence of classifier weights can be extremely slow so that evolution tends to favor piecewise-constant approximations over more-efficient piecewise-linear ones. To improve generalization in XCSF we suggested three new update techniques for speeding up the convergence. One, very simple but limited in generality, consists of using condition predicates to normalize the inputs before the inputs are actually used to calculate and update predictions. The second, with more theoretical guarantees, consists of replacing the original Widrow-Hoff update with the linear least squares (Haykin 1998) technique. The third is the recursive online version of linear least squares.

We performed a number of experiments and statistically analyzed the reported results. Our analysis showed that (i) all the new approaches lead to significantly higher and more effective generalization, and that (ii) least squares approaches appear to be more robust and better performing, although they require more computational

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

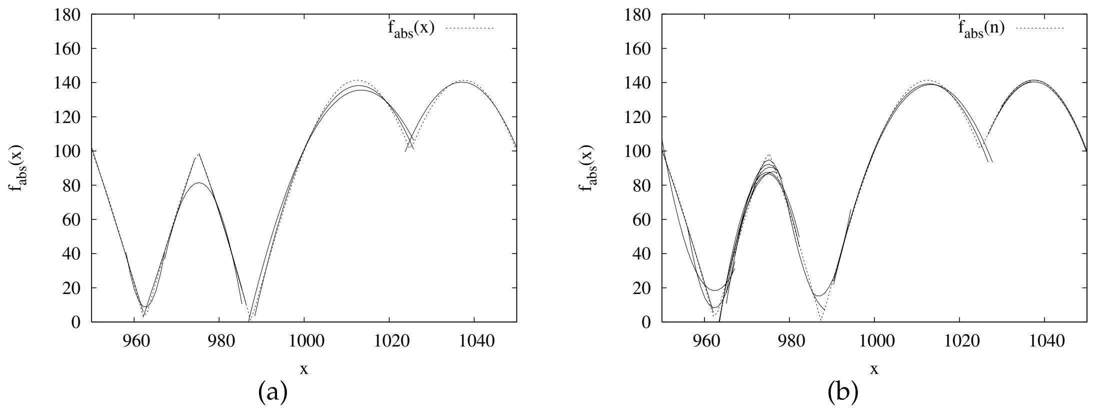


Figure 15: XCSFls with quadratic approximation applied to $f_{abs}(x)$ when $\epsilon_0 = 5$: (a) best population evolved and (b) worst population evolved. Lines identify the approximations provided by classifiers. The corresponding approximations are reported in Figure 14b.

resources both in terms of memory (since classifiers have to maintain extra information about past inputs) and in terms of time (since additional matrix operations are required). Finally, we extended XCSF to allow polynomial approximations and reported experiments showing that XCSF can exploit this basis and evolve correspondingly more compact solutions. Further results regarding XCSF with polynomial approximations are available in (Lanzi et al. 2005a).

Especially with the modifications introduced in this paper, XCSF appears to be a very promising direction for learning classifier systems. Preliminary results on multistep problems (Lanzi et al. 2005e; Lanzi et al. 2005d) as well as on most typical single step problems (Lanzi et al. 2005c) suggest that XCSF can evolve optimal solutions that are generally much more compact than those evolved by other XCS models. Future research directions include: further investigations of XCSF's generalization capabilities, analysis of XCSF as a tool for generalized reinforcement learning, and applications to other domains such as autonomous agents or regression problems.

Acknowledgments

Pier Luca is more than grateful to Xavier Llora, Kumara Sastry, Tian-Li Yu, and all the other members of the Illinois Genetic Algorithm Laboratory for their help and the useful discussions. Pier Luca wishes to thank David and Stewart for invaluable discussions and inspiration. Pier Luca and Daniele wish to thank Marco Colombetti and Stefano Ceri for support. Daniele would like to thank the members of the Artificial Intelligence and Robotics Laboratory of the Politecnico di Milano for the helpful discussions.

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, the National Science Foundation under ITR grant DMR-03-25939 at Materials Computation Center, UIUC. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, ei-

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

ther expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

Bernadó, E., X. Llorà, and J. M. Garrell (2001). XCS and GALE: A comparative study of two learning classifier systems with six other learning algorithms on classification tasks. In *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS-2001)*, pp. 337–34. ACM Press, New-York, NY. Short version published in Genetic and Evolutionary Compution COnference (GECCO2001).

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings 16th International Conference on Machine Learning*, pp. 49–56. Morgan Kaufmann, San Francisco, CA.

Bradtke, S. J. and A. G. Barto (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning 22*(1-3), 33–57.

Butz, M., D. E. Goldberg, and P. L. Lanzi (2003, December). Gradient descent methods in learning classifier systems. Technical Report 2003028, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign.

Butz, M. V., D. E. Goldberg, and P. L. Lanzi (2005, October). Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems. *IEEE Transaction on Evolutionary Computation 9*(5), 452–473.

Butz, M. V. and S. W. Wilson (2001). An algorithmic description of XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson (Eds.), *Advances in Learning Classifier Systems*, Volume 1996 of *LNAI*, pp. 253–272. Berlin: Springer-Verlag.

Butz, M. V. and S. W. Wilson (2002). An algorithmic description of XCS. *Journal of Soft Computing 6*(3–4), 144–153.

Conover, W. J. (1998). *Practical Nonparametric Statistics*. Wiley. third edition.

Dixon, P. W., D. W. Corne, and M. J. Oates (2002). A preliminary investigation of modified XCS as a generic data mining tool. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson (Eds.), *Advances in Learning Classifier Systems*, Volume 2321 of *LNAI*, pp. 133–150. Berlin: Springer-Verlag.

Galassi, M., J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi (2004, July). *GNU Scientific Library Reference Manual* (Free Software Foundation ed.). Edition 1.5, for GSL Version 1.5. Network Theory Ltd. Bristol, UK.

Glantz, S. A. and B. K. Slinker (2001). *Primer of Applied Regression & Analysis of Variance*. McGraw Hill. Second edition.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

Goodwin, G. C. and K. S. Sin (1984). *Adaptive Filtering: Prediction and Control*. Prentice-Hall information and system sciences series.

Haykin, S. (1986). *Adaptive Filter Theory*. Prentice-Hall.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press. Republished by the MIT Press, 1992.

Koza, J. (1992). *Genetic Programming*. MIT Press.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

Lanzi, P. L. (2001). Mining interesting knowledge from data with the XCS classifier system. In L. Spector et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO-2001)*: San Francisco, CA, pp. 958–965. Morgan Kaufmann.

Lanzi, P. L. (2002). The XCS library. `http://xcslib.sourceforge.net`.

Lanzi, P. L., D. Loiacono, S. W. Wilson, and D. E. Goldberg (2005a). Extending XCSF beyond linear approximation. In H.-G. Beyer et al.(Eds.), *GECCO 2005: Genetic and Evolutionary Computation COnference*: Washington DC, pp. 1827–1834. Volume 2, ACM Press.

Lanzi, P. L., D. Loiacono, S. W. Wilson, and D. E. Goldberg (2005b). Generalization in the XCSF classifier system: Analysis, improvement, and extension. Technical Report 2005012, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign. Also available as a technical report of the Dipartimento di Elettronica e Informazione – Politecnico di Milano.

Lanzi, P. L., D. Loiacono, S. W. Wilson, and D. E. Goldberg (2005c, September). XCS with computed prediction for the learning of boolean functions. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, Edinburgh, UK, pp. 588–595. IEEE Press.

Lanzi, P. L., D. Loiacono, S. W. Wilson, and D. E. Goldberg (2005d, September). XCS wIth Computed Prediction In Continuous Multistep Environments. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, Edinburgh, UK, pp. 2032–2039. IEEE Press.

Lanzi, P. L., D. Loiacono, S. W. Wilson, and D. E. Goldberg (2005e). XCS with computed prediction in multistep environments. In H.-G. Beyer et al. (Eds.), *GECCO 2005: Proceedings of the 2005 Genetic and Evolutionary Computation COnference*, Volume 2, Washington DC, pp. 1859–1866. ACM Press.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (Eds.) (2002, February). *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning – An Introduction*. Cambridge, MA: The MIT Press.

Watkins, C. and P. Dayan (1992). Technical note: Q-Learning. *Machine Learning 8*, 279–292.

Widrow, B. and M. E. Hoff (1988). *Adaptive Switching Circuits*, Chapter Neurocomputing: Foundation 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 et al. (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 665–674. Morgan Kaufmann.

Wilson, S. W. (2001a). Function approximation with a classifier system. In L. Spector et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, CA, pp. 974–981. Morgan Kaufmann.

Wilson, S. W. (2001b). Mining oblique data with XCS. Volume 1996 of *Lecture Notes in Computer Science*, pp. 158–174. Springer-Verlag.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021

Wilson, S. W. (2002). Classifiers that approximate functions. *Journal of Natural Computing 1*(2-3), 211–234.

Wilson, S. W. (2004). Classifier systems for continuous payoff environments. In K. Deb et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation COnference – GECCO-2004, Part II*, Volume 3103 of *Lecture Notes in Computer Science*, Seattle, WA, pp. 824–835. Springer-Verlag.

Zelinka, I. (2002). Analytic programming by means of soma algorithm. In *Proceedings of the 8th International Conference on Soft Computing, Mendel'02*, Brno, Czech Republic, pp. 93–101.

Downloaded from http://direct.mit.edu/evco/article-pdf/15/2/133/1493744/evco.2007.15.2.133.pdf by guest on 19 June 2021