Prediction Update Algorithms for XCSF: RLS, Kalman Filter, and Gain Adaptation

Pier Luca Lanzi*†, Daniele Loiacono*, Stewart W. Wilson†‡, David E. Goldberg†
*Artificial Intelligence and Robotics Laboratory (AIRLab)
Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
†Illinois Genetic Algorithm Laboratory (IlliGAL)
University of Illinois at Urbana Champaign, Urbana, IL 61801, USA
‡Prediction Dynamics, Concord, MA 01742, USA
lanzi@elet.polimi.it, loiacono@elet.polimi.it, wilson@prediction-dynamics.com, deg@illigal.ge.uiuc.edu

ABSTRACT

We study how different prediction update algorithms influence the performance of XCSF. We consider three classical parameter estimation algorithms (NLMS, RLS, and Kalman filter) and four gain adaptation algorithms (K1, K2, IDBD, and IDD). The latter have been shown to perform comparably to the best algorithms (RLS and Kalman), but they have a lower complexity. We apply these algorithms to update classifier prediction in XCSF and compare the performances of the seven versions of XCSF on a set of real functions. Our results show that the best known algorithms still perform best: XCSF with RLS and XCSF with Kalman perform significantly better than the others. In contrast, when added to XCSF, gain adaptation algorithms perform comparably to NLMS, the simplest estimation algorithm, the same used in the original XCSF. Nevertheless, algorithms that perform similarly generalize differently. For instance: XCSF with Kalman filter evolves more compact solutions than XCSF with RLS and gain adaptation algorithms allow better generalization than NLMS.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]: Genetics Based Machine Learning; G.1.2 [**Approximation**]: Least squares

General Terms

Algorithms, Performance.

Keywords

LCS, XCS, Prediction Update, Kalman Filter.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

1. INTRODUCTION

In XCSF [14], computed prediction replaces classifier prediction with a parameter vector $\mathbf{w}$, which is associated to each classifier, and a prediction function $p(s, \mathbf{w})$. In XCSF, classifier prediction is *computed* from the current input s and $\mathbf{w}$. In addition, the original update of the prediction parameter is replaced by the update of $\mathbf{w}$; this step depends both on the prediction function $p(s, \mathbf{w})$ and on the parameter estimation algorithm used to update $\mathbf{w}$. In [14], the prediction function $p(s, \mathbf{w})$ is defined as the linear combination "$s\mathbf{w}$" while $\mathbf{w}$ is updated using Normalized Least Mean Square algorithm (NLMS) [6] or modified Delta rule [13, 14].

The results in [14] show that XCSF can evolve accurate piecewise-linear approximations of simple functions. Later, the analysis in [8] has shown that generalization in XCSF relies on the convergence properties of the parameter estimation algorithm used to update $\mathbf{w}$. If the algorithm is too slow, the genetic algorithm may act *before* $\mathbf{w}$ is adequately updated, and the generalization capability of XCSF may be dramatically reduced. In [8], XCSF is applied to a set of periodic functions on different input ranges. The results show that when the input range is limited to small inputs the NLMS update converges quickly so that XCSF evolves classifiers representing piecewise-linear approximations, as in [14]. Instead, when the input range includes large inputs the NLMS update converges slowly so that XCSF evolves piecewise *constant* approximations, which do not fully exploit the generalization potential of XCSF. Accordingly, in [8] NLMS is replaced with recursive least squares (RLS) [6], which is known to be more robust and to converge faster than NLMS. The results in [8] show that XCSF with RLS performs better than the original XCSF with NLMS both in terms of prediction accuracy and in terms of achieved generalization.

In this paper, we extend the analysis in [8] and study how different incremental parameter estimation algorithms influence XCSF performance. We consider (i) two classical parameter estimation algorithm (RLS, and Kalman filter), and (ii) four gain adaptation algorithms (also known as dynamic-learning-rate algorithms [12]), i.e., K1 [12], K2 [12], IDBD [11], and IDD [5]. The first three gain adaptation algorithms (K1, K2, and IDBD) were introduced in [12] as alternatives to the more computationally expensive least squares algorithms. In [12] it was shown, that these methods could perform better than NLMS and comparably to

the best algorithm (Kalman filter in [12]), while having the same order of complexity as NLMS. Later, IDD [5] was introduced as a simplification of IDBD. We extend XCSF and use these six algorithms as a replacement of the usual NLMS prediction update used in XCSF. We compare the seven versions of XCSF both in terms of prediction accuracy and generalization, on three types of problems: one involving the approximation of simple functions (like those used in [14]), one involving the same functions enriched with irrelevant inputs (as done in [12]), one involving Gaussian noise added to the incoming reward (as done in [1]). The results we present show that in XCSF the contribution of the genetic algorithm to the learning process makes some differences among the update algorithms less relevant. The best known algorithms still perform best: XCSF with RLS and XCSF with Kalman perform significantly better than all the others. In contrast, when added to XCSF, gain adaptation algorithms perform comparably to NLMS (in [12], they outperformed NLMS), the simplest estimation algorithm possible, the same used in the original XCSF. Nevertheless, algorithms that perform similarly generalize differently so that for instance: XCSF with Kalman filter evolves more compact solutions than XCSF with RLS; gain adaptation algorithms allow better generalization than NLMS while having the same order of complexity. Finally, our results are coherent with those in [8], RLS performs better than NLMS even in very simple problems.

2. INCREMENTAL PARAMETER ESTIMATION

We consider the typical problem of incrementally estimating the parameter vector $\hat{\boldsymbol{\theta}}(t)$ of the linear system,

$$y(t) = \boldsymbol{\theta}(t)^T \boldsymbol{\phi}(t) + \xi(t) \quad (1)$$

$$\hat{\boldsymbol{\theta}}(t+1) = \hat{\boldsymbol{\theta}}(t) + \mathbf{d}(t) \quad (2)$$

from the observations $\langle \boldsymbol{\phi}(t), y(t) \rangle$, $t \in \{0, 1, \dots\}$; the term $\boldsymbol{\xi}(t)$ is the observation noise and it is assumed to be white noise with zero mean and variance R; the term $\mathbf{d}(t)$ is the *drift* vector, a zero mean white noise with covariance matrix $Q = E\{d(t)d(t)^T\}$; it is usually assumed that $\boldsymbol{\xi}(t)$, $\mathbf{d}(t)$, and $\boldsymbol{\phi}(t)$ are independent random variables [4, 6]. The estimation of $\hat{\boldsymbol{\theta}}(t)$ is usually solved by algorithms which update the current estimate $\boldsymbol{\theta}(t)$ of $\hat{\boldsymbol{\theta}}(t)$ as,

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \mathbf{k}(t)[y(t) - \boldsymbol{\theta}(t)^T \boldsymbol{\phi}(t)] \quad (3)$$

where $\mathbf{k}(t)$ is the *gain* vector whose definition depends on the estimation algorithm applied. In this section we briefly introduce some algorithms showing how they compute the gain vector $\mathbf{k}(t)$.

2.1 Classical Algorithms

Least Mean Square (LMS) is the simplest and probably the most used parameter estimation algorithm. It defines the gain vector $\mathbf{k}(t)$ as

$$\mathbf{k}(t) = \eta \boldsymbol{\phi}(t) \quad (4)$$

where $\eta > 0$ is the *learning rate*. To guarantee the convergence of LMS it is good practice to set the learning rate as [6],

$$\eta = \frac{\bar{\eta}}{E[\boldsymbol{\phi}(t)^T \boldsymbol{\phi}(t)]}$$

where $\bar{\eta} \in (0, 2)$ is the absolute learning rate that is normalized based on the expected value of $\boldsymbol{\phi}(t)^T \boldsymbol{\phi}(t)$, i.e., $E[\boldsymbol{\phi}(t)^T \boldsymbol{\phi}(t)]$.

The Normalized Least Mean Square or NLMS is a modification of LMS that is usually preferred because potentially faster than LMS when large inputs $\boldsymbol{\phi}(t)$ are involved [6]. In NLMS, the gain vector $\mathbf{k}(t)$ is computed as,

$$\mathbf{k}(t) = \frac{\eta \boldsymbol{\phi}(t)}{\boldsymbol{\phi}(t)^T \boldsymbol{\phi}(t)} \quad (5)$$

where the learning rate η is usually bound between 0 and 1 [4].[1] Least squares algorithms compute the gain vector $\mathbf{k}(t)$ as,

$$\mathbf{k}(t) = \frac{\mathbf{P}(t)\boldsymbol{\phi}(t)}{R + \boldsymbol{\phi}(t)^T \mathbf{P}(t) \boldsymbol{\phi}(t)} \quad (6)$$

where, if $\boldsymbol{\theta}(t)$ has n inputs, the $n \times n$ matrix $P(t)$ is recursively updated as,

$$\mathbf{P}(t+1) = \mathbf{P}(t) - \frac{\mathbf{P}(t)\boldsymbol{\phi}(t)\boldsymbol{\phi}(t)^T \mathbf{P}(t)}{R + \boldsymbol{\phi}(t)^T \mathbf{P}(t) \boldsymbol{\phi}(t)} + \mathbf{Q}. \quad (7)$$

When $R = E[\xi^2(t)]$ and $\mathbf{Q} = E[\mathbf{d}(t)\mathbf{d}(t)^T]$, Equation 6 and Equation 7 implement the Kalman filter. In this case, if it is not possible to have an exact estimate of $\xi(t)$ and $\mathbf{d}(t)$, some rough assumptions are made (see [11] further discussion). When $R = I$ and $\mathbf{Q} = 0$, the same equations implement recursive least squares (RLS) [6]. Thus RLS assumes that the observation noise $\boldsymbol{\xi}(t)$ (Equation 1) has unitary variance and that there is no drift $\mathbf{d}(t) = 0$ (Equation 2). We refer the reader to [6] and [4] for an analysis of the relations and similarities shared by recursive least squares and Kalman filter and also for modifications of recursive least squares.

2.2 Gain Adaptation Algorithms

Gain adaptation, or *dynamic-learning-rate*, algorithms have been introduced in [11, 12]. They implement a sort of meta-learning in that the learning rate is adapted based on the current inputs and on the trace of previous modifications. The incremental delta-bar-delta (IDBD) uses a different adaptive learning rate for each input. This approach in principle improves the performance when some inputs are irrelevant or when noise affects the inputs in different ways. With IDBD [11] each element $k_i(t)$ of the gain vector $\mathbf{k}(t)$ is computed as,

$$k_i(t) = e^{\beta_i(t)} \phi_i(t) \quad (8)$$

$$\beta_i(t+1) = \beta_i(t) + \eta \delta(t) \phi_i(t) h_i(t) \quad (9)$$

$$h_i(t+1) = h_i(t) \left[1 - k_i(t)\phi_i(t)\right]^+ + k_i(t)\delta(t) \quad (10)$$

where $\delta(t)$ is the error computed as "$y(t) - \boldsymbol{\theta}(t)^T \boldsymbol{\phi}(t)$", η is the *meta learning rate*, and the function $[x]^+$ returns x if $x > 0$, zero otherwise.

The *incremental delta-delta* (IDD) [5] implements an approach similar to IDBD but it does not store the traces $h_i(t)$. In IDD, each element $k_i(t)$ of the gain vector $\mathbf{k}(t)$ is computed as,

$$k_i(t) = e^{\beta_i(t)} \phi_i(t) \quad (11)$$

$$\beta_i(t+1) = \beta_i(t) + \eta \frac{\Delta\theta_i(t+1)}{e^{\beta_i(t)}} \Delta\theta_i(t) \quad (12)$$

[1] Equation is used in XCSF to update the classifier prediction parameter vector $\mathbf{w}$ [14], see Section 3 for further details.

where $\Delta\theta_i(t)$ is the change of the parameter θ_i which, given the error $\delta(t)$ ($\delta(t) = y(t) - \boldsymbol{\theta}(t)^T\boldsymbol{\phi}(t)$), is computed as $\Delta\theta_i(t) = k_i(t)\delta(t)$; while η is the usual *meta learning rate*.

LMS and NLMS have an order of complexity $O(n)$ while recursive least squares and Kalman filter are $O(n^2)$ both in terms of computation and memory usage. The algorithms K1 and K2 [12] approximate least squares, while being less expensive both in terms of computation and memory. The idea in K1 and K2 is to approximate the matrix $\mathbf{P}$ as a diagonal matrix $\hat{\mathbf{P}}$ so as to reduce the memory usage and the computation from $O(n^2)$ down to $O(n)$, since only the diagonal elements of $\hat{\mathbf{P}}$, $\hat{p_{ii}}$, are stored. In the K1 algorithm [12] this computation is made as,

$$\hat{p}_{ii}(t) = e^{\beta_i(t)} \quad (13)$$

$$\beta_i(t+1) = \beta_i(t) + \eta\delta(t)\phi_i(t)h_i(t) \quad (14)$$

$$h_i(t+1) = [h_i(t) + k_i(t)\delta(t)]\,[1 - k_i(t)\phi_i(t)]^+ \quad (15)$$

where $\delta(t)$ is the error, $\delta(t) = y(t) - \boldsymbol{\theta}(t)^T\boldsymbol{\phi}(t)$, η is the *meta learning rate* and $[x]^+$ is x if $x > 0$, else 0. The algorithm K2 [12] is similar, it still defines $\hat{p}_{ii}(t) = e^{\beta_i(t)}$, but computes the *memory parameters* β_i in a different way,

$$\beta_i(t+1) = \beta_i(t) + \frac{\eta\phi_i^2(t)}{1 + \sum_j \phi_j^4(t)}[\delta^2(t) + -\hat{R} - \sum_j \hat{p}_{jj}(t)\phi_i^2(t)] \quad (16)$$

where $\delta(t)$ is the error, $\delta(t) = y(t) - \boldsymbol{\theta}(t)^T\boldsymbol{\phi}(t)$, η is the *meta learning rate* and $[x]^+$ is x if $x > 0$, else 0. In [12] it is suggested that in practice, it is often useful to bound each β_i from -10, to prevent arithmetic underflows; it is also prudent to limit the change in β_i on any one step to $\pm$ 2. We refer the reader to [12] for more details on the derivation of the algorithms K1 and K2.

3. THE XCSF CLASSIFIER SYSTEM

The introduction of computed prediction requires three simple modifications to XCS. First, the classifier prediction parameter is replaced with a parameter vector $\mathbf{w}$, that is used to compute classifier prediction. Second, a prediction function $p(s_t, \mathbf{w})$ is introduced which defines how classifier prediction is computed based on the current input s_t and $\mathbf{w}$; usually, $p(s_t, \mathbf{w})$ is computed as simple linear combination $s_t\mathbf{w}$ but other functions can be used [9]. Finally, the usual update of the classifier prediction parameter is replaced with the update of the classifier parameter vector $\mathbf{w}$ based on a target prediction value P and the current classifier prediction value $p(s_t, \mathbf{w})$.

Classifiers. In XCSF, classifiers consist of a condition, an action, and four main parameters. The condition is represented by a concatenation of real interval predicates, $int_i = (l_i, u_i)$. The action specifies the action for which the payoff is predicted. The four parameters are: the weight vector $\mathbf{w}$, the prediction error ϵ, the fitness F, and the numerosity *num*. The weight vector $\mathbf{w}$ 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.

Performance Component. At 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 [2]. The weight vector $\mathbf{w}$ of covering classifiers is initialized with zero values while all the other classifiers parameters are initialized as usual [2].

For each action a_i in [M], XCSF computes the *system prediction*. 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. In contrast with XCS, in XCSF classifier prediction is computed as a function of the current state s_t and the classifier weight vector $\mathbf{w}$. Accordingly, in XCSF system prediction is a function of both the current state s_t and the action a. Following a notation similar to that in [2], the system prediction for action a in state s_t, $P(\mathbf{s_t}, a)$, is defined as:

$$P(s_t, a) = \frac{\sum_{cl\in[M]|_a} cl.p(s_t) \times cl.F}{\sum_{cl\in[M]|_a} cl.F} \quad (17)$$

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 in state s_t, which 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. 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.

Reinforcement Component. XCSF uses the incoming reward to update the parameters of classifiers in action set $[A]_{-1}$ corresponding to the previous time step. Note that, when XCSF is used for function approximation (a single step problem) the reinforcement component acts on the current action set. At time step t, the expected payoff P is computed as:

$$P = r_{-1} + \gamma \max_{a\in A} P(\mathbf{s_t}, a) \quad (18)$$

where r_{-1} is the reward received at the previous time step. The expected payoff P is used to update the weight vector $\mathbf{w}$ of the classifier in $[A]_{-1}$ using NLMS or *modified delta rule* [13]. For each classifier $cl \in [A]_{-1}$, each weight $cl.w_i$ is adjusted by a quantity Δw_i computed as:

$$\Delta w_i = \frac{\eta s_{t-1}(i)}{|s_{t-1}(i)|^2}(P - cl.p(\mathbf{s}_{t-1})) \quad (19)$$

where η is the correction rate and $|\mathbf{s}_{t-1}|^2$ is the norm the input vector $\mathbf{s}_{t-1}$ [14]. The values Δw_i are used to update the weights of classifier cl as:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \quad (20)$$

Then the prediction error ϵ is updated as:

$$cl.\epsilon \leftarrow cl.\epsilon + \beta(|P - cl.p(s_t)| - cl.\epsilon) \quad (21)$$

Finally, classifier fitness is updated as in XCS.

Discovery Component. The genetic algorithm is applied as in any other XCS model [14]. The weight vectors of offspring classifiers are set to a fitness weighted average of the parents weight vectors; all the other parameters are initialized as usual [2].

Algorithm 1 Update classifier cl with IDBD algorithm

procedure UPDATE_PREDICTION(cl, s, P)
  $x(0) \leftarrow x_0$; ▷ Build $\mathbf{x}$ by adding x_0 to s
  for $i \in \{1, \ldots, |s|\}$ **do**
    $x(i) \leftarrow s(i)$;
  end for
  $\delta \leftarrow P - cl.p(s)$; ▷ Compute the current error
  for $i \in \{0, \ldots, |s|\}$ **do**
    $\beta(i) \leftarrow \beta(i) + \eta_{meta} \times \delta \times x(i) \times h(i)$; ▷ Update the value of $\beta(i)$
    $\eta \leftarrow \exp(\beta(i))$; ▷ Compute the learning rate for the ith weight
    $cl.w_i \leftarrow cl.w_i + \eta \times \delta \times x(i)$; ▷ Update the value of $cl.w_i$
    $h(i) \leftarrow h(i) \times \text{positive}(1 - \eta \times x(i)^2) + \eta \times \delta \times x(i)$; ▷ Update the value of $h(i)$
  end for
end procedure

4. XCSF WITH LEAST SQUARES AND GAIN ADAPTATION

We extended XCSF and replaced the original NLMS update [14] with each one of the six parameter estimation algorithm discussed in the previous section: recursive least squares, as introduced in [8], Kalman filter [6], K1 [12], K2 [12], IDBD [11], and IDD [5]. These extensions are rather straightforward. The linear system in Equation 1 and the update in Equation 3 easily map into the problem of estimating the parameters $\mathbf{w}$ in XCSF: $y(t)$ corresponds to the classifier prediction, i.e., $p(s, \mathbf{w})$; $\boldsymbol{\phi}(t)$ corresponds to the current input s_t; the estimate $\boldsymbol{\theta}(t)$ of $\hat{\boldsymbol{\theta}}(t)$ corresponds to the classifier parameter vector $\mathbf{w}$. At time t the prediction of a classifier cl (i.e., $y(t)$ in Equation 1) is affected by an error $e(t)$. Thus, the observation noise $\boldsymbol{\xi}(t)$ corresponds to the error $e(t)$ which represents both the error due to environment and the error due to the approximation. Finally, when the environment is stationary (a common assumption in reinforcement learning) there is no drift, i.e., $\mathbf{d}(t)$ is always zero.

For each algorithm considered, we extended the classifier structure by adding the additional parameters that the algorithm requires and we replaced the usual NLMS update [14] with the corresponding update algorithm. To implement the Kalman algorithm in XCSF we needed to estimate the values of R and Q in Equation 7. To estimate R, classifiers are extended with an additional error ϵ_s which estimates the square error of the classifier. The new parameter ϵ_s is updated as,

$$cl.\epsilon_s \leftarrow cl.\epsilon_s + \beta((P - cl.p(s_t))^2 - cl.\epsilon_s)$$

Finally, since all the problems we consider are stationary, there is no drift so that Q is zero.

As an example, Algorithm 1 reports the algorithmic description for the update of classifier cl using IDBD [11]; the algorithmic descriptions for all the six algorithms considered are available in [10]. Table 2 reports for each algorithm the list of parameters that are added to the classifier structure and the corresponding complexity order for memory.

$$f_p(x) = 1 + x + x^2 + x^3 \quad (22)$$
$$f_{abs}(x) = |\sin(x) + |\cos(x)|| \quad (23)$$
$$f_{s3}(x) = \sin(x) + \sin(2x) + \sin(3x) \quad (24)$$
$$f_{s4}(x) = \sin(x) + \sin(2x) + \sin(3x) + sin(4x) \quad (25)$$

Table 1: Test functions ($x \in [0, 1]$).

5. EXPERIMENTAL DESIGN

All the experiments discussed in this paper involve single step problems and are performed following the standard design used in the literature [14]. 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 whom 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 while it is turned off during *test* problems. 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 *root mean square error* (RMSE) defined as:

$$RMSE = \sqrt{\frac{1}{n} \sum_x (f(x) - \hat{f}(x))^2},$$

where n is the number of sample points used to compute the error. In particular, we consider the average RMSE over the performed experiments, dubbed $\overline{RMSE}$.

6. EXPERIMENTAL RESULTS

We have compared the seven versions of XCSF we introduced in the previous section by applying them on the four functions in Table 1 which are the real versions of those used in [7].

6.1 Simple Experiments

In the first set of experiments we applied XCSF to approximate the four functions in Table 1, using the following parameters: $N = 800$; $\beta = 0.2$; $\alpha = 0.1$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 50$, $\theta_{GA} = 50$, and $\delta = 0.1$; GA-subsumption is on with $\theta_{gasub} = 50$; action-set subsumption is off; $m_0 = 0.2$, $r_0 = 0.1$ [14]; we considered three values of ϵ_0, 0.05, 0.1, and 0.2. For XCSF, recursive least squares, and Kalman filter we set $\eta = 0.2$, while for the gain adaptation algorithms we set the meta-learning rate η to 0.02, a rather typical value in [12]. Table 3a reports, for each of the four functions, and each value of ϵ_0, the $\overline{RMSE}$ ($\mu \pm \sigma$) for the seven versions of XCSF: column `xcsf` reports the performance of XCSF which uses the NLMS update [14]; column `rls` reports the performance of the recursive least

squares update; column `kal` reports the performance of the Kalman algorithm (Section 2); columns `k1`, `k2`, and `idbd` report the performance of the three update methods introduced in [12]; finally, column `idd` reports the performance of the IDD algorithm introduced in [5].

XCSF with RLS and Kalman, columns `rls` and `kal`, perform better than all the others which, on the other hand, appear to perform similarly. We applied a *one-way* analysis of variance or ANOVA [3] to test whether the differences in the approximation errors in Table 3a are statistically significant. We also applied the typical post-hoc procedures (SNK, Tukey, Scheffé, and Bonferroni) to analyze the differences among the seven versions of XCSF. The analysis shows that the performances are statistically significant at the 99.99% confidence level; the following post-hoc procedures partition the seven algorithms into four groups with similar (not significantly different) performances. The first group contains RLS and Kalman, columns `rls` and `kal`. The second group contains K1 and IDBD (columns `k1` and `idbd`); this is coherent with the discussion in [12], in fact the two algorithms are related and IDBD can be actually viewed as the immediate ancestor of K1 [12]; the third group contains the usual XCSF update and K2 (columns `xcsf` and `k2`); also this result is coherent with [12], XCSF update is basically an NLMS and K2 is actually an extension of NLMS. Finally, the fourth group contains only IDD which performs differently from all the others.

We analyzed the results for the average population size (Table 3b). For this purpose we considered the groups identified in the previous step and applied an ANOVA in each group to test whether the generalization performance (within the groups) is actually different. The analysis of the first group, containing `rls` and `kal`, shows that the two algorithms perform differently in terms of the size of the evolved populations. Thus, the Kalman algorithm evolves populations that are significantly smaller than those evolved by recursive least squares. The analysis of the second group shows that also `k1` and `idbd` are statistically different in term of average population size, i.e., XCSF with `idbd` evolves populations that are more compact than those evolved by XCSF with `k1`. Finally, the analysis of the third group shows also that the size of the populations evolved by XCSF with NLMS (`xcsf`) and K2 are statistically significant, i.e., K2 evolves populations that are more compact than those evolved by XCSF. These results are confirmed by the statistical analysis of the data for average classifier generality available in [10].

6.2 Adding Random Inputs

In [12], it is shown that gain adaptation algorithms (like K1, K2, and IDBD) perform better than classical parameter identification algorithms (like NLMS, recursive least squares, and Kalman algorithm) in problems involving irrelevant random inputs. Therefore, we extended the previous comparison and, following the approach in [12], we added one and two irrelevant inputs, uniformly randomly generated between 0 and 1, to the four functions in Table 1. The XCSF parameters are set as in the previous experiments, except for the population size N which is 800 when one irrelevant input is added and 1600 when two random inputs are added.

Table 4 reports the $\overline{RMSE}$ and the average population size for the seven versions of XCSF applied to the four functions enriched with one random irrelevant inputs. The one-way ANOVA applied on the data in Table 4a shows that the performances are significantly different. The following post-hoc procedures partition the seven algorithms into three groups. The first group, as in the previous set of experiments, includes RLS and Kalman (`rls` and `kal`) which (according to the tests) perform similarly. The second group includes XCSF, K1, K2, and IDD. The third group includes IDBD which appears to perform significantly worse than all the others. We applied the same statistical test on each of these three groups to test whether the algorithms that performed similarly in terms of prediction error, actually provide different generalization performances. The analysis of the data in Table 4b shows that the generalization performance of the algorithms in the first group is not statistically significant, i.e., RLS and Kalman perform similarly. In the second group the analysis of the average population size, forms three subgroups, one group containing XCSF and K1, one group containing K2, and one group with IDD.

Table 5 reports the $\overline{RMSE}$ for the seven versions of XCSF applied to the four functions enriched with two irrelevant random inputs. The one-way ANOVA applied on the data in Table 5a shows that the performances are significantly different. The following post-hoc procedures partition the seven algorithms into the same three groups found in the case of one irrelevant input. The subsequent analysis of the data regarding the population size (Table 5b) we performed on each group also returned the same result as in the previous analysis.

6.3 Noisy Payoff

Finally, we compared the seven algorithms on the same problems with Gaussian noise added to the reward. Table 6a reports the $\overline{RMSE}$ for the four functions when a zero mean Gaussian noise and a standard deviation σ of 0.05 and 0.1 is added to the reward. We performed two one-way ANOVA, one for each value of σ. Both tests show a statistically significant difference in the performances. When $\sigma = 0.05$, the post-hoc procedures partition the seven algorithms into four groups: one with RLS and Kalman (`rls` and `kal`); one with the NLMS and K2; one with K1 only; one with IDBD and IDD. We partitioned the data in Table 6a for $\sigma = 0.05$ according to the groups identified by the previous ANOVA and applied three separate analyzes on each group to determine whether there is a statistically significant difference within the algorithms in each group. The analysis shows that (i) the difference in the average population size between RLS and Kalman (`rls` and `kal`) is statistically significant, thus XCSF with Kalman tends to evolve more compact solutions; (ii) the difference in the average population size between NLMS and K2 (`xcsf` and `k2`) is statistically significant; (iii) the difference in the average population size between IDBD and IDD (`idbd` and `idd`) is statistically significant. When $\sigma = 0.1$, the post-hoc procedures group together NLMS, K1, and K2 (the other two groups are the same as in the case of $\sigma = 0.05$); thus, with more noise the difference between NLMS, K1, and K2 appears to be less relevant. The subsequent analysis of the average population size within the three groups shows that: (i) the difference in the average population size between RLS and Kalman is significant, i.e., XCSF with Kalman evolves

Algorithm	Parameters	Complexity
rls	$\mathbf{k}(t)$, $\mathbf{P}$	$O(n^2)$
kal	$\mathbf{k}(t)$, $\mathbf{P}$, $\mathbf{Q}$, ε_s	$O(n^2)$
k1	$\mathbf{k}(t)$, $\boldsymbol{\beta}(t), \mathbf{h}(t)$	$O(n)$
k2	$\mathbf{k}(t)$, $\boldsymbol{\beta}(t)$	$O(n)$
idbd	$\mathbf{k}(t)$, $\boldsymbol{\beta}(t), \mathbf{h}(t)$	$O(n)$
idd	$\mathbf{k}(t)$, $\boldsymbol{\beta}(t)$	$O(n)$

Table 2: Additional parameters and complexity.

more compact populations; (ii) the difference in the average population size between NLMS, K1, and K2 is significant and, coherently to the analysis performed for $\sigma = 0.05$, it groups together NLMS with K2 which have according to analysis similar performance when compared to K1; (iii) the difference in the average population size for IDBD and IDD is not statistically significant. Overall, these results confirm what has been found in the previous experiments: (i) RLS and Kalman always perform better than NLMS and gain adaptation algorithms, although Kalman appears to allow better generalization than RLS; (ii) in terms of prediction accuracy, NLMS performs rather well and in many case its performance is comparable to more sophisticated gain adaptation algorithms, (iii) however, gain adaptation algorithms seem to allow better generalization than NLMS.

7. CONCLUSIONS

We have studied the influence of different prediction update algorithms on XCSF. The results we report show that the genetic pressure toward accurate maximally general classifiers flatten some of the differences that have been reported in the parameter estimation literature. The additional complexity required by RLS and Kalman filter may be convenient: XCSF with RLS and XCSF with Kalman always perform significantly better than all the other versions. Gain adaptation algorithms may perform better than NLMS, one of the simplest parameter estimation algorithm, but the difference appear to be less evident than that reported in [12]. This is due to the presence of the additional evolutionary pressure that is present in XCSF but not in the typical settings like those in [12]. Nevertheless, algorithms with similar learning performance provide different achieved generalization. In fact, our results show that XCSF with Kalman filter usually evolves more compact solutions than XCSF with RLS, although the two versions achieve similar performance. Finally, gain adaptation algorithms usually allow better generalization than NLMS, although providing similar performance.

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129. 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, either expressed or implied, of the Air Force Office of Scientific Research, or the U.S. Government.

The authors also wish to thank Rich Sutton for his help with IDBD.

8. REFERENCES

[1] M. V. Butz, K. Sastry, and D. E. Goldberg. Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, 6(1):53–77, 2005.

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

[3] S. A. Glantz and B. K. Slinker. *Primer of Applied Regression & Analysis of Variance*. McGraw Hill, 2001. second edition.

[4] G. C. Goodwin and K. S. Sin. *Adaptive Filtering: Prediction and Control*. Prentice-Hall information and system sciences series, Mar. 1984.

[5] M. E. Harmon and L. C. B. III. Multi-player residual advantage learning with general function. Technical report, Air Force Base Ohio: Wright Laboratory, 1996.

[6] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, 2001. 4th Edition.

[7] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Extending XCSF beyond linear approximation. In *Genetic and Evolutionary Computation – GECCO-2005*, pages 1859–1866, Washington DC, USA, 2005. ACM Press.

[8] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Generalization in the XCSF classifier system: Analysis, improvement, and extension. Technical Report 2005012, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign, 2005.

[9] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. XCS with computed prediction for the learning of Boolean functions. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, pages 588–595, Edinburgh, UK, Sept. 2005. IEEE.

[10] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Prediction update algorithms for XCSF: Rls, kalman filter, and gain adaptation. Technical Report 2006008, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign, 2006.

[11] R. S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 171–176. MIT Press, 1992.

[12] R. S. Sutton. Gain adaptation beats least squares? In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pages 161–166. Yale University, New Haven, CT, 1992.

[13] B. Widrow and M. E. Hoff. *Adaptive Switching Circuits*, chapter Neurocomputing: Foundation of Research, pages 126–134. The MIT Press, Cambridge, 1988.

[14] S. W. Wilson. Classifiers that approximate functions. *Journal of Natural Computating*, 1(2-3):211–234, 2002.

$f(x)$	ϵ_0	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.05	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.02 ± 0.00
$f_p(x)$	0.1	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.03 ± 0.01	0.04 ± 0.00	0.02 ± 0.00	0.04 ± 0.00
$f_p(x)$	0.2	0.04 ± 0.01	0.04 ± 0.01	0.06 ± 0.01	0.05 ± 0.01	0.07 ± 0.01	0.04 ± 0.01	0.07 ± 0.01
$f_{s3}(x)$	0.05	0.10 ± 0.05	0.03 ± 0.01	0.03 ± 0.00	0.08 ± 0.02	0.09 ± 0.04	0.08 ± 0.02	0.13 ± 0.04
$f_{s3}(x)$	0.1	0.10 ± 0.02	0.05 ± 0.00	0.05 ± 0.01	0.10 ± 0.03	0.09 ± 0.01	0.09 ± 0.02	0.12 ± 0.03
$f_{s3}(x)$	0.2	0.13 ± 0.01	0.09 ± 0.01	0.09 ± 0.01	0.12 ± 0.01	0.13 ± 0.01	0.11 ± 0.01	0.15 ± 0.02
$f_{s4}(x)$	0.05	0.16 ± 0.06	0.04 ± 0.03	0.03 ± 0.01	0.12 ± 0.04	0.15 ± 0.05	0.13 ± 0.03	0.17 ± 0.05
$f_{s4}(x)$	0.1	0.16 ± 0.06	0.06 ± 0.01	0.06 ± 0.01	0.13 ± 0.04	0.13 ± 0.03	0.12 ± 0.02	0.16 ± 0.03
$f_{s4}(x)$	0.2	0.17 ± 0.03	0.09 ± 0.01	0.09 ± 0.01	0.15 ± 0.02	0.16 ± 0.01	0.14 ± 0.02	0.18 ± 0.03
$f_{abs}(x)$	0.05	0.04 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.04 ± 0.01	0.04 ± 0.00	0.04 ± 0.01	0.05 ± 0.01
$f_{abs}(x)$	0.1	0.05 ± 0.01	0.04 ± 0.01	0.04 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.05 ± 0.01	0.07 ± 0.00
$f_{abs}(x)$	0.2	0.10 ± 0.01	0.09 ± 0.01	0.08 ± 0.01	0.10 ± 0.01	0.13 ± 0.01	0.08 ± 0.01	0.12 ± 0.01

(a)

$f(x)$	ϵ_0	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.05	93.98 ± 8.97	128.10 ± 9.26	55.52 ± 5.50	99.78 ± 10.91	85.54 ± 8.20	100.76 ± 8.30	82.88 ± 7.45
$f_p(x)$	0.1	115.36 ± 9.73	128.70 ± 8.88	49.50 ± 5.72	107.20 ± 10.48	93.08 ± 8.63	117.36 ± 8.72	60.14 ± 6.37
$f_p(x)$	0.2	126.14 ± 9.21	123.08 ± 8.66	43.12 ± 5.99	120.30 ± 11.31	117.76 ± 9.92	119.98 ± 10.24	50.40 ± 7.29
$f_{s3}(x)$	0.05	128.84 ± 9.45	89.08 ± 7.80	80.00 ± 7.02	121.74 ± 10.07	126.46 ± 8.47	103.70 ± 6.75	130.36 ± 9.65
$f_{s3}(x)$	0.1	108.82 ± 9.45	86.28 ± 7.95	69.80 ± 7.10	104.66 ± 9.83	107.18 ± 6.27	85.92 ± 8.59	109.90 ± 7.11
$f_{s3}(x)$	0.2	95.16 ± 8.05	95.82 ± 9.62	63.08 ± 6.26	92.02 ± 6.50	93.96 ± 7.40	77.72 ± 7.78	87.60 ± 7.05
$f_{s4}(x)$	0.05	141.92 ± 7.71	91.08 ± 7.19	87.22 ± 7.57	135.82 ± 8.65	138.38 ± 8.03	118.42 ± 8.23	133.42 ± 7.34
$f_{s4}(x)$	0.1	122.80 ± 9.05	88.00 ± 8.83	75.78 ± 5.90	112.03 ± 9.06	113.28 ± 9.56	96.70 ± 9.37	115.40 ± 7.50
$f_{s4}(x)$	0.2	101.90 ± 6.91	91.22 ± 9.42	68.10 ± 6.31	95.60 ± 8.79	96.82 ± 9.25	85.45 ± 7.57	95.60 ± 6.73
$f_{abs}(x)$	0.05	102.44 ± 10.84	86.68 ± 10.00	63.54 ± 5.79	104.96 ± 9.20	107.36 ± 7.94	77.62 ± 10.41	106.50 ± 7.85
$f_{abs}(x)$	0.1	97.30 ± 8.49	98.40 ± 9.59	62.78 ± 6.28	96.22 ± 8.39	95.72 ± 6.93	76.22 ± 6.18	81.80 ± 7.20
$f_{abs}(x)$	0.2	102.20 ± 9.21	112.84 ± 9.42	57.94 ± 6.20	96.74 ± 9.63	92.12 ± 10.57	89.34 ± 8.49	59.76 ± 5.79

(b)

Table 3: Performance of XCSF: (a) $\overline{RMSE}$, (b) average population size.

$f(x)$	ϵ_0	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.05	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.03 ± 0.00	0.02 ± 0.00	0.02 ± 0.00
$f_p(x)$	0.1	0.03 ± 0.00	0.02 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.04 ± 0.01	0.03 ± 0.01	0.04 ± 0.01
$f_p(x)$	0.2	0.05 ± 0.01	0.06 ± 0.01	0.04 ± 0.00	0.05 ± 0.01	0.08 ± 0.01	0.04 ± 0.01	0.08 ± 0.01
$f_{s3}(x)$	0.05	0.14 ± 0.03	0.03 ± 0.01	0.03 ± 0.01	0.13 ± 0.02	0.13 ± 0.03	0.18 ± 0.03	0.15 ± 0.03
$f_{s3}(x)$	0.1	0.12 ± 0.02	0.05 ± 0.00	0.05 ± 0.00	0.12 ± 0.02	0.11 ± 0.02	0.17 ± 0.03	0.13 ± 0.02
$f_{s3}(x)$	0.2	0.14 ± 0.01	0.09 ± 0.01	0.09 ± 0.00	0.13 ± 0.01	0.14 ± 0.01	0.15 ± 0.02	0.15 ± 0.01
$f_{s4}(x)$	0.05	0.18 ± 0.04	0.05 ± 0.03	0.07 ± 0.05	0.18 ± 0.02	0.17 ± 0.04	0.25 ± 0.04	0.19 ± 0.03
$f_{s4}(x)$	0.1	0.17 ± 0.03	0.06 ± 0.01	0.06 ± 0.00	0.18 ± 0.03	0.15 ± 0.03	0.23 ± 0.03	0.18 ± 0.03
$f_{s4}(x)$	0.2	0.18 ± 0.02	0.09 ± 0.01	0.10 ± 0.01	0.18 ± 0.02	0.17 ± 0.02	0.22 ± 0.03	0.19 ± 0.02
$f_{abs}(x)$	0.05	0.05 ± 0.01	0.02 ± 0.00	0.02 ± 0.00	0.05 ± 0.01	0.05 ± 0.01	0.08 ± 0.01	0.05 ± 0.01
$f_{abs}(x)$	0.1	0.06 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.06 ± 0.00	0.07 ± 0.00	0.08 ± 0.02	0.07 ± 0.01
$f_{abs}(x)$	0.2	0.12 ± 0.00	0.09 ± 0.00	0.08 ± 0.01	0.11 ± 0.00	0.12 ± 0.00	0.11 ± 0.01	0.13 ± 0.00

(a)

$f(x)$	ϵ_0	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.05	152.30 ± 17.34	213.10 ± 11.72	213.54 ± 12.36	157.70 ± 15.66	120.40 ± 13.49	136.04 ± 10.92	124.52 ± 15.86
$f_p(x)$	0.1	178.62 ± 12.53	217.50 ± 11.57	219.40 ± 12.49	190.84 ± 12.64	115.02 ± 11.35	165.78 ± 10.66	108.96 ± 9.48
$f_p(x)$	0.2	210.54 ± 9.83	216.70 ± 8.73	219.16 ± 10.27	212.48 ± 12.63	105.80 ± 8.85	195.54 ± 13.40	102.14 ± 8.68
$f_{s3}(x)$	0.05	236.48 ± 12.69	148.80 ± 11.47	145.60 ± 9.72	240.44 ± 15.44	205.30 ± 11.69	220.32 ± 15.31	239.44 ± 12.18
$f_{s3}(x)$	0.1	188.06 ± 11.56	159.50 ± 10.03	153.64 ± 11.40	189.46 ± 14.33	168.52 ± 12.60	189.80 ± 18.55	185.70 ± 11.07
$f_{s3}(x)$	0.2	161.16 ± 12.02	173.28 ± 11.23	172.50 ± 10.50	159.22 ± 12.74	150.50 ± 12.60	154.44 ± 14.64	143.80 ± 10.98
$f_{s4}(x)$	0.05	250.76 ± 12.68	142.56 ± 13.58	152.56 ± 13.04	251.14 ± 14.93	228.62 ± 12.28	241.04 ± 15.62	247.10 ± 14.42
$f_{s4}(x)$	0.1	208.02 ± 12.60	148.32 ± 11.31	148.34 ± 13.94	212.96 ± 12.80	193.64 ± 12.00	200.74 ± 17.41	207.16 ± 12.76
$f_{s4}(x)$	0.2	173.22 ± 11.89	163.12 ± 8.52	162.38 ± 11.17	174.94 ± 15.42	163.54 ± 11.69	166.78 ± 15.67	165.14 ± 10.23
$f_{abs}(x)$	0.05	173.56 ± 12.07	165.04 ± 13.21	161.26 ± 12.64	178.90 ± 11.48	154.98 ± 11.80	181.98 ± 18.25	166.00 ± 11.63
$f_{abs}(x)$	0.1	157.82 ± 12.82	181.20 ± 11.30	173.22 ± 9.99	155.22 ± 11.14	128.56 ± 7.58	147.40 ± 14.11	154.08 ± 13.33
$f_{abs}(x)$	0.2	185.58 ± 12.95	206.82 ± 12.30	200.98 ± 13.26	184.44 ± 11.67	124.76 ± 9.09	132.46 ± 9.75	120.22 ± 9.60

(b)

Table 4: Performance of XCSF with one irrelevant input: (a) $\overline{RMSE}$, (b) average population size.

$f(x)$	ϵ_0	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.05	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.04 ± 0.05	0.03 ± 0.01
$f_p(x)$	0.1	0.03 ± 0.01	0.03 ± 0.01	0.03 ± 0.01	0.03 ± 0.01	0.04 ± 0.01	0.03 ± 0.01	0.04 ± 0.01
$f_p(x)$	0.2	0.07 ± 0.02	0.06 ± 0.01	0.05 ± 0.00	0.06 ± 0.01	0.09 ± 0.02	0.04 ± 0.01	0.08 ± 0.02
$f_{s3}(x)$	0.05	0.16 ± 0.03	0.05 ± 0.03	0.05 ± 0.04	0.18 ± 0.03	0.13 ± 0.07	0.29 ± 0.06	0.18 ± 0.03
$f_{s3}(x)$	0.1	0.16 ± 0.02	0.06 ± 0.00	0.01 ± 0.03	0.17 ± 0.03	0.13 ± 0.06	0.29 ± 0.06	0.17 ± 0.03
$f_{s3}(x)$	0.2	0.17 ± 0.01	0.09 ± 0.01	0.09 ± 0.01	0.17 ± 0.02	0.16 ± 0.02	0.27 ± 0.05	0.18 ± 0.01
$f_{s3}(x)$	0.05	0.10 ± 0.02	0.03 ± 0.01	0.03 ± 0.01	0.08 ± 0.02	0.10 ± 0.03	0.09 ± 0.02	0.14 ± 0.03
$f_{s3}(x)$	0.1	0.11 ± 0.02	0.05 ± 0.00	0.05 ± 0.00	0.12 ± 0.02	0.11 ± 0.02	0.20 ± 0.08	0.12 ± 0.02
$f_{s3}(x)$	0.2	0.13 ± 0.01	0.08 ± 0.01	0.08 ± 0.01	0.13 ± 0.01	0.13 ± 0.01	0.19 ± 0.05	0.15 ± 0.01
$f_{abs}(x)$	0.05	0.05 ± 0.01	0.02 ± 0.00	0.02 ± 0.00	0.04 ± 0.00	0.04 ± 0.01	0.15 ± 0.06	0.06 ± 0.01
$f_{abs}(x)$	0.1	0.06 ± 0.00	0.04 ± 0.00	0.03 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.12 ± 0.04	0.08 ± 0.01
$f_{abs}(x)$	0.2	0.12 ± 0.00	0.09 ± 0.00	0.09 ± 0.01	0.11 ± 0.00	0.13 ± 0.01	0.12 ± 0.02	0.13 ± 0.01

(a)

$f(x)$	ϵ_0	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.05	440.60 ± 36.37	560.46 ± 26.01	556.94 ± 35.26	473.74 ± 35.91	311.64 ± 28.54	322.10 ± 38.96	330.14 ± 66.68
$f_p(x)$	0.1	506.64 ± 31.14	581.24 ± 28.14	570.26 ± 28.03	527.84 ± 28.17	301.18 ± 29.25	361.76 ± 44.68	295.78 ± 30.85
$f_p(x)$	0.2	571.12 ± 26.36	596.50 ± 21.48	596.02 ± 25.08	568.56 ± 30.11	304.74 ± 28.31	481.62 ± 26.50	275.14 ± 21.40
$f_{s3}(x)$	0.05	128.84 ± 9.45	89.08 ± 7.80	96.64 ± 7.36	121.74 ± 10.07	126.46 ± 8.47	103.70 ± 6.75	132.62 ± 9.49
$f_{s3}(x)$	0.1	522.40 ± 29.56	430.22 ± 25.95	426.32 ± 24.03	544.80 ± 28.40	602.82 ± 221.86	532.58 ± 133.96	497.68 ± 36.60
$f_{s3}(x)$	0.2	453.02 ± 31.68	471.86 ± 24.43	463.54 ± 26.56	461.02 ± 31.08	601.68 ± 262.76	410.46 ± 88.02	374.06 ± 27.43
$f_{s4}(x)$	0.05	666.30 ± 31.94	399.24 ± 26.27	429.35 ± 29.01	666.00 ± 27.52	638.98 ± 127.55	609.62 ± 46.51	663.14 ± 34.34
$f_{s4}(x)$	0.1	568.12 ± 30.74	399.16 ± 30.89	410.60 ± 18.70	580.38 ± 30.55	585.08 ± 154.60	543.16 ± 55.35	536.86 ± 32.35
$f_{s4}(x)$	0.2	490.98 ± 28.21	436.70 ± 27.81	424.98 ± 23.78	486.04 ± 32.99	444.16 ± 30.86	446.24 ± 38.16	435.38 ± 22.96
$f_{abs}(x)$	0.05	489.50 ± 31.28	441.84 ± 21.73	432.04 ± 26.26	488.38 ± 29.06	404.50 ± 27.75	502.64 ± 53.48	450.94 ± 35.98
$f_{abs}(x)$	0.1	426.16 ± 29.00	485.50 ± 30.47	464.90 ± 28.80	432.46 ± 26.09	325.06 ± 24.96	384.76 ± 46.36	329.76 ± 28.85
$f_{abs}(x)$	0.2	506.60 ± 30.74	547.80 ± 29.66	532.78 ± 28.70	494.62 ± 32.33	309.94 ± 24.93	333.46 ± 25.71	302.50 ± 30.34

(b)

Table 5: Performance of XCSF with two irrelevant inputs: (a) $\overline{RMSE}$, (b) average population size.

$f(x)$	ϵ_0	σ	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.1	0.05	0.06 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.01	0.06 ± 0.00
$f_p(x)$	0.2	0.05	0.07 ± 0.01	0.06 ± 0.00	0.07 ± 0.01	0.06 ± 0.01	0.09 ± 0.00	0.07 ± 0.01	0.08 ± 0.01
$f_{s3}(x)$	0.1	0.05	0.10 ± 0.02	0.05 ± 0.00	0.05 ± 0.01	0.09 ± 0.02	0.09 ± 0.03	0.13 ± 0.04	0.12 ± 0.03
$f_{s3}(x)$	0.2	0.05	0.13 ± 0.01	0.09 ± 0.01	0.09 ± 0.01	0.11 ± 0.01	0.13 ± 0.01	0.15 ± 0.01	0.15 ± 0.02
$f_{s4}(x)$	0.1	0.05	0.14 ± 0.03	0.06 ± 0.00	0.06 ± 0.00	0.14 ± 0.03	0.12 ± 0.02	0.16 ± 0.03	0.16 ± 0.04
$f_{s4}(x)$	0.2	0.05	0.17 ± 0.03	0.10 ± 0.01	0.09 ± 0.01	0.14 ± 0.02	0.15 ± 0.02	0.19 ± 0.05	0.19 ± 0.03
$f_{abs}(x)$	0.1	0.05	0.07 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.07 ± 0.01	0.08 ± 0.00	0.07 ± 0.01	0.08 ± 0.00
$f_{abs}(x)$	0.2	0.05	0.11 ± 0.01	0.09 ± 0.00	0.10 ± 0.01	0.10 ± 0.01	0.13 ± 0.01	0.10 ± 0.01	0.13 ± 0.01
$f_p(x)$	0.1	0.1	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.11 ± 0.01	0.10 ± 0.00	0.11 ± 0.00	0.10 ± 0.00
$f_p(x)$	0.2	0.1	0.11 ± 0.00	0.10 ± 0.00	0.11 ± 0.00	0.11 ± 0.01	0.11 ± 0.00	0.11 ± 0.01	0.11 ± 0.00
$f_{s3}(x)$	0.1	0.1	0.14 ± 0.01	0.10 ± 0.00	0.10 ± 0.00	0.14 ± 0.01	0.13 ± 0.02	0.15 ± 0.02	0.15 ± 0.03
$f_{s3}(x)$	0.2	0.1	0.15 ± 0.01	0.13 ± 0.01	0.12 ± 0.01	0.15 ± 0.01	0.15 ± 0.01	0.17 ± 0.01	0.18 ± 0.02
$f_{s4}(x)$	0.1	0.1	0.17 ± 0.02	0.11 ± 0.00	0.11 ± 0.01	0.16 ± 0.01	0.15 ± 0.02	0.19 ± 0.03	0.18 ± 0.03
$f_{s4}(x)$	0.2	0.1	0.19 ± 0.02	0.13 ± 0.01	0.13 ± 0.01	0.18 ± 0.02	0.18 ± 0.01	0.20 ± 0.02	0.20 ± 0.02
$f_{abs}(x)$	0.1	0.1	0.11 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.12 ± 0.01	0.11 ± 0.00	0.12 ± 0.01	0.11 ± 0.00
$f_{abs}(x)$	0.2	0.1	0.09 ± 0.01	0.07 ± 0.01	0.09 ± 0.01	0.09 ± 0.01	0.11 ± 0.01	0.08 ± 0.01	0.12 ± 0.01

(a)

$f(x)$	ϵ_0	σ	xcsf	rls	kal	k1	k2	idbd	idd
$f_p(x)$	0.1	0.05	121.02 ± 10.00	130.26 ± 8.13	51.16 ± 6.49	130.04 ± 8.91	99.94 ± 8.68	104.52 ± 7.82	64.10 ± 6.87
$f_p(x)$	0.2	0.05	124.48 ± 9.96	121.74 ± 11.40	45.26 ± 5.04	122.88 ± 11.23	114.50 ± 10.86	119.28 ± 10.17	48.74 ± 4.97
$f_{s3}(x)$	0.1	0.05	106.24 ± 9.43	68.64 ± 6.63	69.80 ± 7.10	94.32 ± 8.19	101.88 ± 8.26	108.74 ± 8.47	109.90 ± 7.11
$f_{s3}(x)$	0.2	0.05	83.20 ± 7.35	61.96 ± 4.80	63.08 ± 6.26	74.26 ± 8.26	76.80 ± 6.62	85.74 ± 7.21	87.60 ± 7.05
$f_{s4}(x)$	0.1	0.05	119.26 ± 8.47	76.20 ± 6.74	77.96 ± 6.91	110.14 ± 8.43	113.72 ± 7.67	114.62 ± 8.00	116.02 ± 6.92
$f_{s4}(x)$	0.2	0.05	94.36 ± 7.84	70.22 ± 6.03	69.80 ± 6.13	87.46 ± 7.48	87.00 ± 8.32	96.44 ± 7.83	95.76 ± 6.69
$f_{abs}(x)$	0.1	0.05	100.40 ± 8.72	100.72 ± 7.82	62.74 ± 6.14	98.76 ± 7.88	101.58 ± 7.42	73.34 ± 7.34	91.82 ± 7.61
$f_{abs}(x)$	0.2	0.05	102.54 ± 8.98	113.56 ± 11.02	58.50 ± 5.23	105.88 ± 9.99	91.48 ± 8.59	86.00 ± 5.96	59.30 ± 6.94
$f_p(x)$	0.1	0.1	147.90 ± 9.47	156.14 ± 8.86	86.66 ± 6.01	145.48 ± 7.83	120.92 ± 9.07	83.14 ± 5.25	96.72 ± 9.49
$f_p(x)$	0.2	0.1	128.78 ± 10.38	127.34 ± 9.63	47.38 ± 6.51	127.60 ± 9.70	118.38 ± 10.81	118.62 ± 8.40	52.86 ± 6.16
$f_{s3}(x)$	0.1	0.1	135.34 ± 9.32	105.70 ± 6.86	107.70 ± 8.15	133.86 ± 10.64	125.96 ± 7.95	132.88 ± 7.15	132.60 ± 7.53
$f_{s3}(x)$	0.2	0.1	88.70 ± 7.27	65.92 ± 7.12	68.00 ± 5.41	97.36 ± 9.88	84.30 ± 6.63	94.22 ± 7.39	95.14 ± 6.10
$f_{s4}(x)$	0.1	0.1	142.22 ± 8.62	107.70 ± 8.08	111.20 ± 6.20	140.04 ± 10.39	132.92 ± 8.64	137.86 ± 9.02	135.76 ± 9.14
$f_{s4}(x)$	0.2	0.1	100.84 ± 6.52	72.74 ± 5.61	72.68 ± 6.62	94.50 ± 7.83	96.02 ± 6.42	103.02 ± 6.87	102.74 ± 8.03
$f_{abs}(x)$	0.1	0.1	136.14 ± 8.19	129.98 ± 10.24	97.82 ± 8.42	135.54 ± 9.50	122.74 ± 8.27	74.60 ± 5.98	127.54 ± 8.02
$f_{abs}(x)$	0.2	0.1	107.08 ± 8.31	116.10 ± 10.07	56.72 ± 5.16	104.42 ± 9.35	102.32 ± 9.65	86.54 ± 7.84	58.36 ± 6.14

(b)

Table 6: Performance of XCSF with Gaussian noise on the reward: (a) $\overline{RMSE}$, (b) average population size.