Recursive Least Squares and Quadratic Prediction in Continuous Multistep Problems

Daniele Loiacono[†] and Pier Luca Lanzi[†‡]
[†]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

ABSTRACT

XCS with computed prediction, namely XCSF, has been recently extended in several ways. In particular, a novel prediction update algorithm based on recursive least squares and the extension to polynomial prediction led to significant improvements of XCSF. However, these extensions have been studied so far only on single step problems and it is currently not clear if these findings might be extended also to multistep problems. In this paper we investigate this issue by analyzing the performance of XCSF with recursive least squares and with quadratic prediction on continuous multistep problems. Our results show that both these extensions improve the convergence speed of XCSF toward an optimal performance. As showed by the analysis reported in this paper, these improvements are due to the capabilities of recursive least squares and of polynomial prediction to provide a more accurate approximation of the problem value function after the first few learning problems.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning

General Terms

algorithms performance

Keywords

learning classifier systems , recursive least squares, multistep problems

1. INTRODUCTION

Learning Classifier Systems are a genetic based machine learning technique for solving problems through the interaction with an unknown environment. The XCS classifier system [16] is probably the most successful learning classifier system to date. It couples effective temporal difference learning, implemented as a modification of the well-known Q-learning [14], to a niched genetic algorithm guided by an accuracy based fitness to evolve accurate maximally general solutions. In [18] Wilson extended XCS with the idea of *computed prediction* to improve the estimation of the classifiers prediction. In XCS with computed prediction, XCSF in brief, the classifier prediction is not memorized into a parameter but computed as a linear combination of the current input and a weight vector associated to each classifier. Recently, in [11] the classifier weights update has been improved with a recursive least squares approach and the idea of computed prediction has been further extended to polynomial prediction. Both the recursive least squares update and the polynomial prediction have been effectively applied to solve function approximation problems as well as to learn Boolean functions. However, so far it is not currently clear whether these findings might be extended also to continuous multistep problems, where Wilson's XCSF has been already successfully applied [9]. In this paper we investigate this important issue. First, we extend the recursive least squares update algorithm to multistep problems with the *covariance resetting*, a well known approach to deal with a non stationary target. Then, to test our approach, we compare the usual Widrow-Hoff update rule to the recursive least squares (extended with covariance resetting) on a class of continuous multistep problems, the 2D Gridworld problems [1]. Our results show that XCSF with recursive least squares outperforms XCSF with Widrow-Hoff rule in terms of convergence speed, although both reach finally an optimal performance. Thus, the results confirm the findings of previous works on XCSF with recursive least squares applied to single step problems. In addition, we performed a similar experimental analysis to investigate the effect of polynomial prediction on the same set of problems. Also in this case, the results suggest that quadratic prediction results in a faster convergence of XCSF toward the optimal performance. Finally, to explain why recursive least squares and polynomial prediction increase the convergence speed of XCSF we showed that they improve the accuracy of the payoff landscape learned in the first few learning problems.

2. XCS WITH COMPUTED PREDICTION

XCSF differs from XCS in three respects: (i) classifier conditions are extended for numerical inputs, as done in XCSI [17]; (ii) classifiers are extended with a vector of weights w, that are used to compute prediction; finally, (iii) the original update of classifier prediction must be modified so that the weights are updated instead of the classifier prediction. These three modifications result in a version of XCS, XCSF [18, 19], that maps numerical inputs into actions with an associated calculated prediction. In the original paper [18] classifiers have no action and it is assumed that XCSF outputs the estimated prediction, instead of the action itself. In this paper, we consider the version of XCSF with actions and linear prediction (named XCS-LP [19]) in which more than one action is available. As said before, throughout the paper we do not keep the (rather historical) distinction between XCSF and XCS-LP since the two systems are basically identical except for the use of actions in the latter case.

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 [17], 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 specifies the action for which the payoff is predicted. The four parameters are: the weight vector w, used to compute the classifier prediction as a function of the current input; the prediction error ε, that estimates the error affecting classifier prediction; the fitness F that estimates the accuracy of the classifier prediction; the numerosity *num*, a counter used to represent different copies of the same classifier. Note that the size of the weight vector w depends on the type of approximation. In the case of piecewise-linear approximation, considered in this paper, the weight vector 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. XCSF works as 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 $l_i = s_t(i) - \text{rand}(r_0)$, and $u_i = s_t(i) + \text{rand}(r_0)$, where $s_t(i)$ is the input value of state s_t matched by the interval predicated int_i, and the function $\text{rand}(r_0)$ generates a random integer in the interval $[0, r_0]$ with r_0 fixed integer. The weight vector w of covering classifiers is randomly initialized with values from [-1,1]; all the other parameters are initialized as in XCS (see [3]).

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 classifier prediction is computed as a function of the current state s_t and the classifier vector weight w. Accordingly, in XCSF system prediction is a function of both the current state s and the action a. Following a notation similar to [2], the system prediction for action a in state s_t, $P(s_t, a)$, is defined as:

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

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, when piecewise-linear approximation is considered, $cl.p(s_t)$ 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 P is returned to the system.

Reinforcement Component. XCSF uses the incoming reward P to update the parameters of classifiers in action set [A]. The weight vector w of the classifiers in [A] is updated using a *modified delta rule* [15]. For each classifier $cl \in [A]$, each weight $cl.w_i$ is adjusted by a quantity Δw_i computed as:

$$\Delta w_i = \frac{\eta}{|s_t(i)|^2}(P - cl.p(s_t))s_t(i) \quad (2)$$

where η is the correction rate and $|s_t|^2$ is the norm the input vector s_t, (see [18] for details). Equation 2 is usually referred to as the "*normalized*" Widrow-Hoff update or "*modified* delta rule", because of the presence of the term $|\mathbf{s}_t(i)|^2$ [5]. The values Δw_i are used to update the weights of classifier cl as:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \quad (3)$$

Then the prediction error ε is updated as:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|P - cl.p(s_t)| - cl.\varepsilon)$$

Finally, classifier fitness is updated as in XCS.

Discovery Component. The genetic algorithm and subsumption deletion in XCSF work as in XCSI [17]. On a regular basis depending on the parameter θ_{ga}, the genetic algorithm is applied to classifiers in [A]. It selects two classifiers with probability *proportional to their fitness*, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. Crossover and mutation work as in XCSI [17, 18]. The resulting offspring are inserted into the population and two classifiers are deleted to keep the population size constant.

3. IMPROVING AND EXTENDING COMPUTED PREDICTION

The idea of computed prediction, introduced by Wilson in [18], has been recently improved and extended in several ways [11, 12, 6, 10]. In particular, Lanzi et al. extended the computed prediction to polynomial functions [7] and they introduced in [11] a novel prediction update algorithm, based on the recursive least squares. Although these extensions proved to be very effective in single step problems, both in function approximation problems [11, 7] and in boolean problems [8], they have never been applied to multistep problems so far. In the following, we briefly describe the classifier update algorithm based on recursive least squares and how it can be applied to multistep problems. Finally, we show how computed prediction can be extended to polynomial prediction.

3.1 XCSF with Recursive Least Squares

In XCSF with recursive least squares,the Widrow-Hoff rule used to update the classifier weights is replaced with a more effective update algorithm based on recursive least squares (RLS). At time step t, given the current state $\mathbf{s}_t$ and the target payoff P, recursive least squares update the weight vector $\mathbf{w}$ as

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

where $\mathbf{x}_t = [x_0 \quad \mathbf{s}_t]^T$ and $\mathbf{k}_t$, called *gain vector*, is computed as

$$\mathbf{k}_t = \frac{V_{t-1}\mathbf{x}_t}{1 + \mathbf{x}_t^T V_{t-1}\mathbf{x}_t}, \tag{4}$$

while matrix $\mathbf{V}_t$ is computed recursively by,

$$\mathbf{V}_t = \left[I - \mathbf{k}_t\mathbf{x}_t^T\right] V_{t-1}. \tag{5}$$

The matrix $\mathbf{V}(t)$ is usually initialized as $\mathbf{V}(0) = \delta_{rls}\mathbf{I}$, where δ_{rls} is a positive constant and $\mathbf{I}$ is the $n \times n$ identity matrix. An higher δ_{rls}, denotes that initial parametrization is uncertain, accordingly, initially the algorithm will use a higher, thus faster, update rate ($\mathbf{k}_t$). A lower δ_{rls}, denotes that initial parametrization is rather certain, accordingly the algorithm will use a slower update. It is worthwhile to say that the recursive least squares approach presented above involves two basic underlying assumptions [5, 4]: (i) the noise on the target payoff P used for updating the classifier weights can be modeled as a unitary variance white noise and (ii) the optimal classifier weights vector does not change during the learning process, i.e., the problem is stationary. While the first assumption is often reasonable and has usually a small impact on the final outcome, the second assumption is not justified in many problems and may have a big impact on the performance. In the literature [5, 4] many approaches have been introduced for relaxing this assumption. In particular, a straightforward approach is the *resetting* of the matrix $\mathbf{V}$: every τ_{rls} updates, the matrix $\mathbf{V}$ is reset to its initial value $\delta_{rls}\mathbf{I}$. Intuitively, this prevent RLS to converge toward a fixed parameter estimate by continually restarting the learning process. We refer the interested reader to [5, 4] for a more detailed analysis of recursive least squares and other related approaches, like the well known Kalman filter. The extension of XCSF with recursive least squares is straightforward: we added to each classifier the matrix V as an additional parameter and we replaced the usual update of classifier weights with the recursive least squares update described above and reported as Algorithm 1.

3.2 Beyond Linear Prediction

Usually in XCSF the classifier prediction is computed as a linear function, so that piecewise linear approximations of the action-value function are evolved. However, XCSF can be easily extended to evolve also polynomial approximations. Let us consider a simple problem with a single variable state space. At time step t, the classifier prediction is computed as,

$$cl.p(s_t) = w_0x_0 + w_1s_t,$$

where x_0 is a constant input and s_t is the current state. Thus, we can introduce a quadratic term in the approximation evolved by XCSF:

$$cl.p(s_t) = w_0x_0 + w_1s_t + w_2s_t^2. \tag{6}$$

Algorithm 1 Update classifier *cl* with RLS algorithm

```
 1: procedure UPDATE_PREDICTION($cl$, $s$, $P$)
 2:     error ← $P - cl.p(s)$;          ▷ Compute the current error
 3:     $x(0) \leftarrow x_0$;          ▷ Build $\mathbf{x}$ by adding $x_0$ to $s$
 4:     for $i \in \{1, \ldots, |s|\}$ do
 5:         $x(i) \leftarrow s(i)$;
 6:     end for
 7:     if # of updates from last reset > $\tau_{rls}$ then
 8:         $cl.V \leftarrow \delta_{rls}\mathbf{I}$          ▷ Reset $cl$.V
 9:     end if
10:     $\eta_{rls} \leftarrow (1 + \mathbf{x} \cdot cl.V \cdot \mathbf{x}^T)^{-1}$;
11:     $cl.V \leftarrow cl.V - \eta_{rls}^{-1} cl.V \cdot \mathbf{x}^T\mathbf{x} \cdot cl.V$;   ▷ Update $cl.V$
12:     $\mathbf{k}^T \leftarrow cl.V \cdot \mathbf{x}^T$;
13:     for $i \in \{0, \ldots, |s|\}$ do   ▷ Update classifier's weights
14:         $cl.w_i \leftarrow cl.w_i + \mathbf{k}(i) \cdot$ error;
15:     end for
16: end procedure
```

To learn the new set of weights we use the usual XCSF update algorithm (e.g., either RLS or Widrow-Hoff) applied to the input vector $\mathbf{x}_t$, defined as $\mathbf{x}_t = \langle x_0, s_t, s_t^2\rangle$. When more variables are involved, so that $\mathbf{s}_t = \langle s_t(1), \ldots, s_t(n)\rangle$, we define

$$\mathbf{x}_t = \langle x_0, s_t(1), s_t^2(1), \ldots, s_t(n), s_t^2(n)\rangle,$$

and apply XCSF to the newly defined input space. The same approach can be generalized to allow the approximation of any polynomials of order k by extending the input vector $\mathbf{x}_t$ with high order terms. However in this paper, for the sake of simplicity, we will limit our analysis to the quadratic prediction.

4. EXPERIMENTAL DESIGN

To study how the recursive least squares and the quadratic prediction affect the performance of XCSF on continuous multistep problems we considered a well known class of problems: the 2D gridworld problems, introduced in [1]. They are two dimensional environments in which the current state is defined by a pair of real valued coordinates $\langle x, y\rangle$ in $[0, 1]^2$, the only goal is in position $\langle 1, 1\rangle$, and there are four possible actions (left, right, up, and down) coded with two bits; each action corresponds in a step of size s in the corresponding direction; actions that would take the system outside the domain $[0, 1]^2$ take the system to the nearest position of the grid border. The system can start *anywhere* but in the goal position and it reaches the goal position when *both* coordinates are equal or greater than one. When the system reaches the goal it receives 0, in all the other cases it receives -0.5. We called the problem described above *empty gridworld*, dubbed `Grid(s)`, where s is the agent step size. Figure 1a shows the optimal value function associated to the empty gridworld problem, when $s = 0.05$ and $\gamma = 0.95$.

A slightly more challenging problem can be obtained by adding some obstacles to the empty gridworld environment, as proposed in [1]: each obstacle represents an area in which there is an additional cost for moving. These areas are called "puddles" [1], since they actually create a sort of puddle in the optimal value function. Figure 1b depicts the `Puddles(s)` environment that is derived from `Grid(s)` by adding two puddles (the gray areas). When the system is in a puddle, it receives an additional negative reward of -

2, i.e., the action has an additional cost of -2; in the area where the two puddles overlap, the darker gray region, the two negative rewards add up, i.e., the action has a total additional cost of -4. We called this second problem *puddle world*, dubbed `Puddles(s)`, where s is the agent step size. Figure 1c shows the optimal value function of the puddle world, when $s = 0.05$ and $\gamma = 0.95$.

The performance is computed as the average number of steps to reach the goal during the last 100 test problems. To speed up the experiments, problems can last at most 500 steps; when this limit is reached the problem stops even if the system did not reach the goal. All the statistics reported in this paper are averaged over 20 experiments.

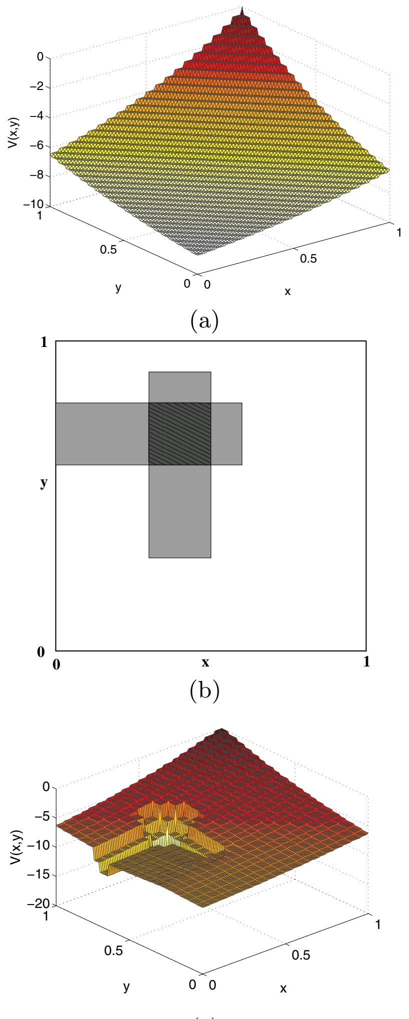


Figure 1: The 2D Continuous Gridworld problems: (a) the optimal value function of `Grid`(0.05) when γ=0.95; (b) the `Puddles`(0.05) environment; (c) the optimal value function of `Puddles`(0.05) when γ=0.95.

5. EXPERIMENTAL RESULTS

Our aim is to study how the RLS update and the quadratic prediction affect the performance of XCSF on continuous multistep problems. To this purpose we applied XCSF with different type of prediction, i.e., linear and quadratic, and with different update rules, i.e., Widrow-Hoff and RLS, on the `Grid`(0.05) and `Puddles`(0.05) problems. In addition, we also compared the performance of XCSF to the one obtained with the tabular Q-learning [13], a standard reference in the RL literature. In order to apply tabular Q-learning to the 2D Gridworld problems, we discretized the the continuous problem space, using the step size $s = 0.05$ as resolution for the discretization process. In the first set of experiments we investigated the effect of the RLS update on the performance of XCSF, while in the second set of experiments we extended our analysis also to quadratic prediction. Finally, we analyzed the results obtained and the accuracy of the action-value approximations learned by the different versions of XCSF.

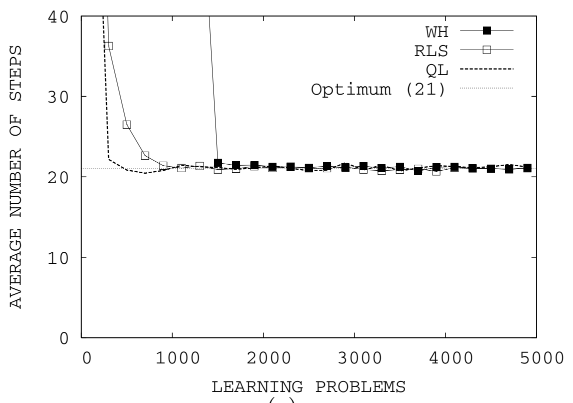
5.1 Results with Recursive Least Squares

In the first set of experiments we compared the Q-learning and XCSF with the two different updates on the 2D continuous gridworld problems. For XCSF we used the following parameter settings: $N = 5000$, $\epsilon_0 = 0.05$; $\beta = 0.2$; $\alpha = 0.1$; $\gamma = 0.95$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $p_{\text{explr}} = 0.5$, $\theta_{del} = 50$, $\theta_{GA} = 50$, and $\delta = 0.1$; GA-subsumption is on with $\theta_{sub} = 50$; while action-set subsumption is off; the parameters for integer conditions are $m_0 = 0.5$, $r_0 = 0.25$ [17]; the parameter x_0 for XCSF is 1 [18]. In addition, with the RLS update we used $\delta_{rls} = 10$ and $\tau_{rls} = 50$. Accordingly, for Q-learning we set $\beta = 0.2$, $\gamma = 0.95$, and $p_{\text{explr}} = 0.5$. The Figure 2a compares the performance of Q-learning and of the two versions of XCSF on the `Grid`(0.05) problem. All the systems are able to reach an optimal performance and XCSF with the RLS update is able to learn much faster than XCSF with the Widrow-Hoff update, although Q-learning is even faster. This is not surprising, as Q-learning is provided with the optimal state space discretization to solve the problem, while XCSF has to search for it. However it is worthwhile to notice that when the RLS update rule is used, XCSF is able to learn almost as fast as the Q-learning. Moving to the more difficult `Puddles`(0.05) problem, we find very similar results as showed by Figure 2b. Also in this case, XCSF with RLS update is able to learn faster than XCSF with the usual Widrow-Hoff update rule and the difference with Q-learning is even less evident.

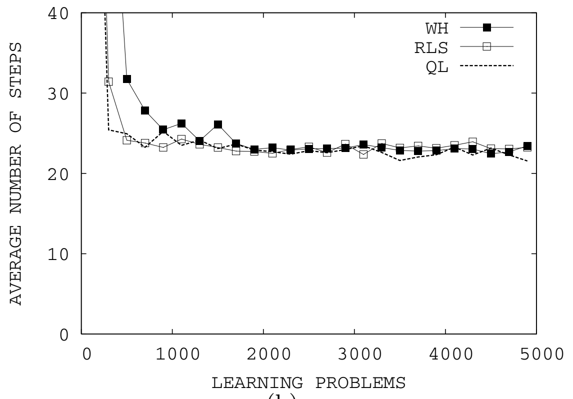
Therefore, our results suggest that the RLS update rule is able to exploit the experience collected more effectively than the Widrow-Hoff rule and confirm the previous findings on single step problems reported in [11].

5.2 Results with Quadratic Prediction

In the second set of experiments, we compared linear prediction to quadratic prediction on the `Grid`(0.05) and the `Puddles`(0.05) problems, using both Widrow-Hoff and RLS updates. Parameters are set as in the previous experiments. Table 1a reports the performance of the systems in the first 500 test problems as a measure of the convergence speed. As found in the previous set of experiments, the RLS update leads to a faster convergence, also when quadratic prediction is used. In addition, the results suggest that also quadratic prediction affects the learning speed: both with Widrow-Hoff update and with the RLS update the quadratic prediction outperforms the linear one. In particular, XCSF with the quadratic prediction and the RLS update is able to learn even faster than Q-learning in both `Grid`(0.05) and `Puddles`(0.05) problems. However, as Table 1b shows, all



(a)



(b)

Figure 2: The performance of Q-learning (reported as QL), XCSF with the Widrow-Hoff update (reported as WH), and of XCSF with the RLS update (reported as RLS) applied to: (a) Grid(0.05) problem (b) Puddles(0.05) problem. Curves are averages on 20 runs.

System	Grid(0.05)	Puddles(0.05)
Q-learning	27.87 ± 0.84	32.12 ± 1.03
XCSF with LINEAR WH	97.75 ± 136.75	89.13 ± 22.36
XCSF with LINEAR RLS	38.46 ± 16.28	35.20 ± 7.60
XCSF with QUADRATIC WH	33.08 ± 6.07	62.56 ± 14.77
XCSF with QUADRATIC RLS	25.75 ± 3.11	29.51 ± 2.97

(a)

System	Grid(0.05)	Puddles(0.05)
Q-learning	21.21 ± 0.18	22.64 ± 0.16
XCSF with LINEAR WH	21.10 ± 0.32	22.90 ± 0.46
XCSF with LINEAR RLS	20.99 ± 0.28	23.15 ± 0.73
XCSF with QUADRATIC WH	20.99 ± 0.21	22.78 ± 0.38
XCSF with QUADRATIC RLS	20.97 ± 0.38	22.96 ± 0.56

(b)

System	Grid(0.05)	Puddles(0.05)
XCSF with LINEAR WH	1231.10 ± 47.04	1491.45 ± 38.17
XCSF with LINEAR RLS	1526.55 ± 29.61	1599.95 ± 40.25
XCSF with QUADRATIC WH	1401.45 ± 35.07	1494.95 ± 44.35
XCSF with QUADRATIC RLS	1577.60 ± 44.23	1596.70 ± 44.15

(c)

Table 1: XCSF applied to Grid(0.05) and to Puddles(0.05) problems. (a) Average number of steps to reach the goal per episode in the first 500 test problems; (b) average number of steps to reach the goal per episode in the last 500 test problems; (c) size of the population evolved. Statistics are averages over 20 experiments.

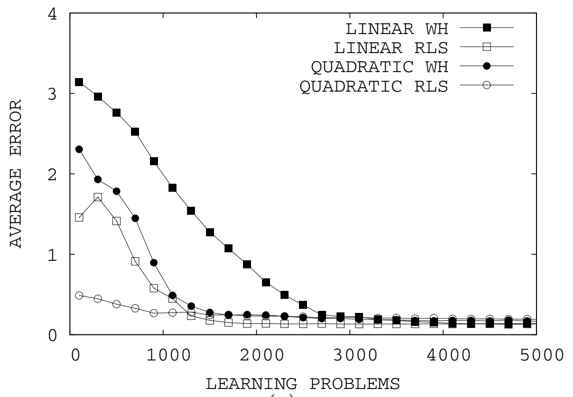
the systems reach an optimal performance. Finally, it can be noticed that the number of macroclassifiers evolved (Table 1c) is very similar for all the systems, suggesting that XCSF with quadratic prediction does not evolve a more compact solution.

5.3 Analysis of Results

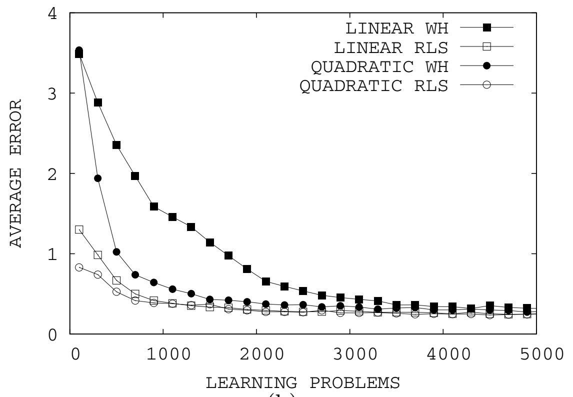
Our results suggest that in continuous multistep problems, the RLS update and the quadratic prediction does not give any advantage either in terms of final performance or in terms of population size. On the other hand, both these extensions lead to an effective improvement of the learning speed, that is they play an important role in the early stage of the learning process. However, this results is not surprising: (i) the RLS update exploits more effectively the experience collected and learns faster an accurate approximation; (ii) the quadratic prediction allows a broader generalization in the early stages that leads very quickly to a rough approximation of the payoff landscape. Figure 3 reports the error of the value function learned by the four XCSF versions during the learning process. The error of a learned value function is measured as the absolute error with respect to the optimal value function, computed as the average of the absolute errors over an uniform grid of 100×100 samples of the problem space. For each version of XCSF this error measure is computed at different stages of the learning process and then averaged over the 20 runs to generate the error curves reported in Figure 3. Results confirm our hypothesis: both quadratic prediction and RLS update lead very fast to accurate approximations of the optimal value function, although the final approximations are as accurate as the one evolved by XCSF with Widrow-Hoff rule and linear prediction. To better understand how the different versions of XCSF approximate the value function, Figure 4, Figure 5, Figure 6, and Figure 7 show some examples of the value functions learned by XCSF at different stages of the learning process. In particular, Figure 4a and Figure 5a show the value function learned by XCSF with linear prediction after few learning episodes, using respectively the Widrow-Hoff update and the RLS update. While the value function learned by XCSF with Widrow-Hoff is flat and very uninformative, the one learned by XCSF with RLS update provides a rough approximation to the slope of the optimal value function, despite it is still far from being accurate. Finally, Figure 6 and Figure 7 report similar examples of value functions learned by XCSF with quadratic predictions. Figure 7a shows how XCSF with both quadratic prediction and RLS update may learn very quickly a rough approximations of the optimal value function after very few learning episodes. A similar analysis can be performed on the Puddles(0.05) but it is not reported here due to the lack of spaces.

6. CONCLUSIONS

In this paper we investigated the application of two successful extensions of XCSF, the recursive least squares update algorithm and the quadratic prediction, to multistep problems First, we extended the recursive least squares approach, originally devised only for single step problems, to

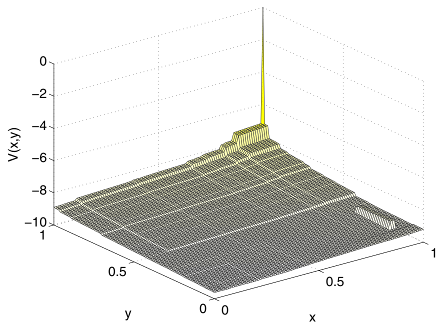


(a)

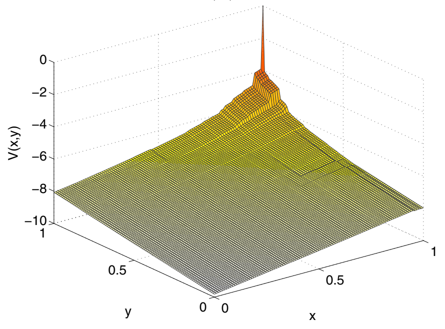


(b)

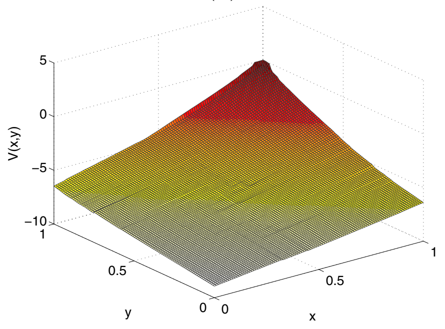
Figure 3: Average absolute error of the value functions learned by XCSF on (a) the `Grid`(0.05) problem and (b) the `Puddles`(0.05) problem. Curves are averages over 20 runs.



(a)



(b)



(c)

Figure 4: Examples of the value function evolved by XCSF with linear prediction and Widrow-Hoff update on the `Grid`(0.05) problem: (a) after 50 learning episodes (b) after 500 learning episodes (c) at the end of the experiment (after 5000 learning episode).

the multistep problems with the *covariance resetting*, a technique to deal with a non stationary target. Second, we showed how the linear prediction used by XCSF can be extended to quadratic prediction in a very straightforward way. Then the recursive least squares update and the quadratic prediction have been compared to the usual XCSF on the 2D Gridworld problems. Our results suggest that the recursive least squares update as well as the quadratic prediction lead to a faster convergence speed of XCSF toward the optimal performance. The analysis of the accuracy of the value function estimate showed that recursive least squares and quadratic prediction plays an important role in the early stage of the learning process. The capabilities of recursive least squares of exploiting more effectively the experience collected and the broader generalization allowed by the quadratic prediction, lead to a more accurate estimate of the value function after few learning episode. In conclusion, we showed that the previous findings on recursive least squares and polynomial prediction applied to single step problems can be extended also to continuous multistep problems. Further investigations will include the analysis of the generalizations evolved by XCSF with recursive least squares and quadratic prediction.

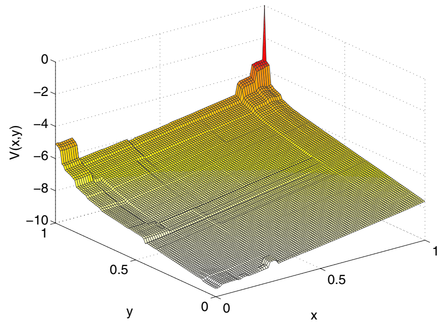
7. REFERENCES

[1] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pages 369–376, Cambridge, MA, 1995. The MIT Press.

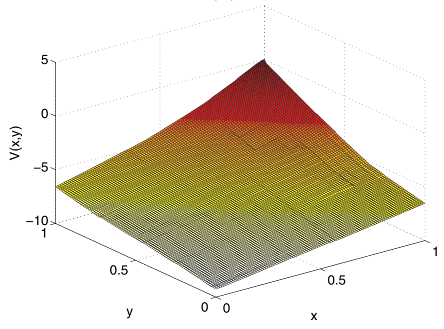
[2] Martin V. Butz and Martin Pelikan. Analyzing the evolutionary pressures in XCS. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 935–942, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

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

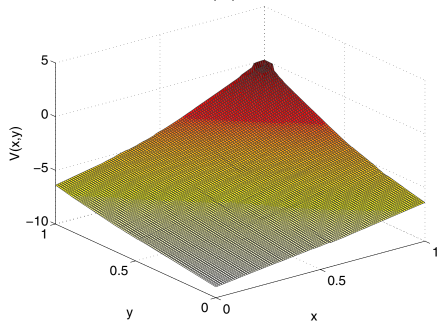
[4] Graham C. Goodwin and Kwai Sang Sin. *Adaptive Filtering: Prediction and Control.* Prentice-Hall information and system sciences series, March 1984.



(a)

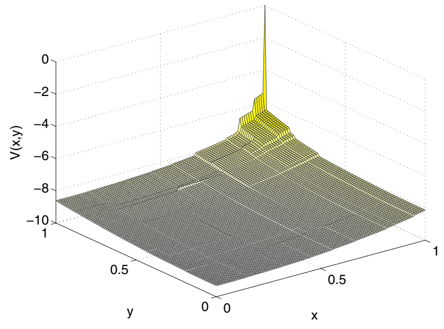


(b)

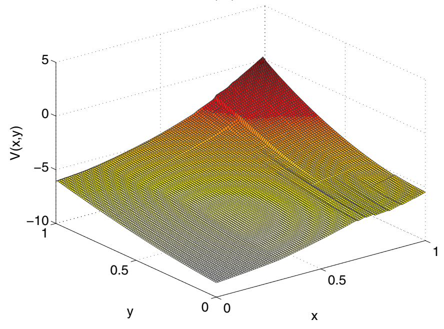


(c)

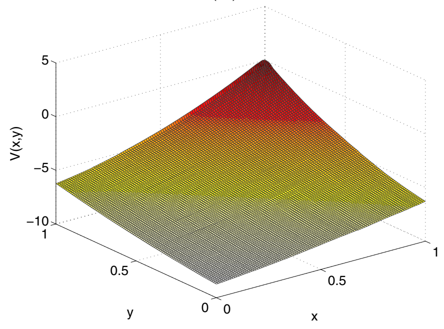
Figure 5: Examples of the value function evolved by XCSF with linear prediction and RLS update on the `Grid`(0.05) problem: (a) after 50 learning episodes (b) after 500 learning episodes (c) at the end of the experiment (after 5000 learning episode).



(a)



(b)



(c)

Figure 6: Examples of the value function evolved by XCSF with quadratic prediction and Widrow-Hoff update on the `Grid`(0.05) problem: (a) after 50 learning episodes (b) after 500 learning episodes (c) at the end of the experiment (after 5000 learning episode).

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

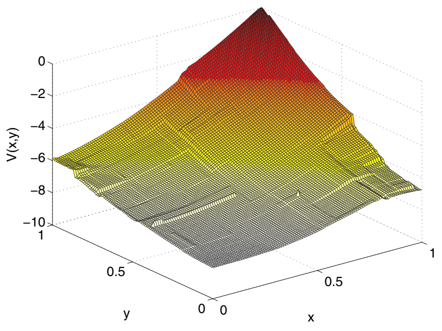
[6] P. L. Lanzi and D. Loiacono. XCSF with neural prediction. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2270–2276, 2006.

[7] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Extending XCSF beyond linear approximation. In *Genetic and Evolutionary Computation – GECCO-2005*, pages 1859–1866, Washington DC, USA, 2005. ACM Press.

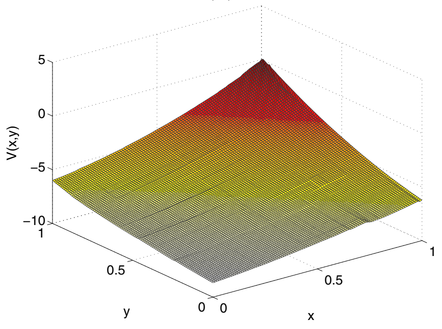
[8] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with computed prediction for the learning of boolean functions. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, pages 588–595, Edinburgh, UK, September 2005. IEEE.

[9] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with computed prediction in continuous multistep environments. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, pages 2032–2039, Edinburgh, UK, September 2005. IEEE.

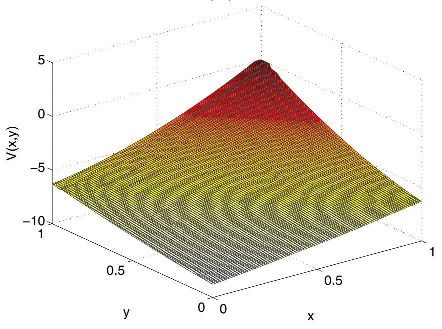
[10] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Prediction update algorithms for XCSF: RLS, kalman filter, and gain adaptation. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1505–1512, New York, NY, USA, 2006. ACM Press.



(a)



(b)



(c)

Figure 7: Examples of the value function evolved by XCSF with quadratic prediction and RLS update on the `Grid`(0.05) problem: (a) after 50 learning episodes (b) after 500 learning episodes (c) at the end of the experiment (after 5000 learning episode).

[11] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Generalization in the XCSF classifier system: Analysis, improvement, and extension. *Evolutionary Computation*, 15(2):133–168, 2007.

[12] Daniele Loiacono, Andrea Marelli, and Pier Luca Lanzi. Support vector regression for classifier prediction. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1806–1813, New York, NY, USA, 2007. ACM Press.

[13] C.J.C.H. Watkins. *Learning from delayed reward*. PhD thesis, 1989.

[14] C.J.C.H. Watkins and P. Dayan. Technical note: *Q*-Learning. *Machine Learning*, 8:279–292, 1992.

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

[16] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

[17] Stewart W. Wilson. Mining Oblique Data with XCS. In *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000), in the Joint Workshops of SAB 2000 and PPSN 2000*, pages 158–174, 2000. Pier Luca Lanzi, Wolfgang Stolzmann and Stewart W. Wilson (workshop organisers).

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

[19] Stewart W. Wilson. Classifier systems for continuous payoff environments. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 824–835, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.