Stochastic Decision Making in Learning Classifier Systems through a Natural Policy Gradient Method

Gang Chen, Mengjie Zhang, Shaoning Pang, and Colin Douch

Victoria University of Wellington,
Unitec Institute of Technology, New Zealand
{aaron.chen,mengjie.zhang}@ecs.vuw.ac.nz,
ppang@unitec.ac.nz,
douchcoli@myvuw.ac.nz

Abstract. Learning classifier systems (LCSs) are rule-based machine learning technologies designed to learn optimal decision-making policies in the form of a compact set of maximally general and accurate rules. A study of the literature reveals that most of the existing LCSs focused primarily on learning *deterministic* policies. However a desirable policy may often be *stochastic*, in particular when the environment is partially observable. To fill this gap, based on XCS, which is one of the most successful accuracy-based LCSs, a new Michigan-style LCS called Natural XCS (i.e. NXCS) is proposed in this paper. NXCS enables direct learning of stochastic policies by utilizing a natural gradient learning technology under a policy gradient framework. Its effectiveness is experimentally compared with XCS and one of its variation known as XCS_μ in this paper. Our results show that NXCS can achieve competitive performance in both deterministic and stochastic multi-step problems.

1 Introduction

Originated from John Holland's seminal work on cognitive systems [5,6], learning classifier systems (LCSs) are rule-based machine learning technologies designed to learn optimal decision-making policies in the form of a compact set of maximally general and accurate rules (aka. classifiers) [13]. Among all LCSs developed to date, XCS, which was introduced by Wilson, is unarguably the most successful accuracy-based LCS [8]. In this paper, a new Michigan-style LCS with native support for stochastic decision making will be developed based on XCS.

LCSs have been successfully applied to solve *reinforcement learning* problems where a learning agent is situated in a multi-step environment often modeled as a Markov Decision Process (MDP) [11]. A study of the literature reveals that reinforcement learning is commonly conducted in LCSs by approximating the *state-action value function*, which is represented jointly by a group of classifiers.

Due to the value-function based approach, the aim of a LCS is to learn *deterministic* policies. However, learning *stochastic* policies is often shown to be more *reliable*, in particular when the environment is stochastic or *partially observable* [12]. To the best of our knowledge, few LCSs have ever attempted to

C.K. Loo et al. (Eds.): ICONIP 2014, Part III, LNCS 8836, pp. 300–307, 2014.

directly learn stochastic policies that explicitly associate with each action a suitable probability for it to be performed in every state.

In view of the gap in the literature, a new LCS called Natural XCS (i.e. NXCS) will be developed in this paper. NXCS enables direct learning of stochastic policies by utilizing a *natural gradient learning technology* under a *policy gradient framework* [1]. Inspired by several temporal-difference based natural learning algorithms [2, 10], this paper presents the first study of natural gradient learning in LCSs.

The remainder of this paper is organized as follows. A short introduction to the XCS classifier system can be found in Section 2. Based on XCS, NXCS will be further developed in Section 3. The performance of NXCS is experimentally compared with XCS and its recent variation known as XCS$_\mu$ [9] in Section 4. Finally Section 5 concludes this paper.

2 XCS Classifier System

XCS is an effective reinforcement learning method in which generalization is obtained through evolving a *population* $[P]$ of classifiers. A detailed algorithmic description of XCS can be found in [4]. At any discrete time t, a learning agent receives sensory inputs from the current environment state s_t. It reacts by performing an action a chosen from A. The environment then transits to a new state at $t+1$, i.e. s_{t+1}, and a reward r_{t+1} is provided as feedback to the agent. The goal of the agent is to maximize the amount of reward obtained in the long run. We briefly review the four key components of XCS in this section.

Classifier: In XCS, each classifier cl has a condition c_{cl}, an action a_{cl}, and several other parameters, including 1) the prediction p_{cl} that estimates the average payoff upon using the classifier; 2) the prediction error ϵ_{cl} ; and 3) the fitness F_{cl} that estimates the average relative accuracy of classifier cl.

Performance Component: Whenever a decision is to be made at any time t, XCS creates a *match set* $[M]_t$ containing all classifiers in the population that match the current sensory input from state s_t. For every action $a \in A$, the agent calculates the predicted value of performing a, i.e. $P_t(a)$, based on the prediction from every classifier belonging to $[M]_t$ [4]. After that, an action a will be selected and the corresponding group of classifiers recommending a will form the *action set* at time t, i.e. $[A]_t$. The selected action will then be performed and the reward r_{t+1} will be received subsequently. During learning, the ϵ-*greedy* selection method will be exploited to randomize action selection. During testing, however, an exploitation strategy will be employed such that the action a with the highest $P_t(a)$ will always be selected.

Reinforcement Component: Upon reaching a new state s_{t+1}, the parameters of those classifiers in $[A]_t$ will be updated according to [4]. In particular, the prediction p_{cl} of a classifier $cl \in [A]_t$ will be updated based on:

$$p_{cl}(t+1) \leftarrow p_{cl}(t) + \beta \left(r_{t+1} + \gamma \max_{a \in A} P_{t+1}(a) - p_{cl}(t) \right) \tag{1}$$

where β is a fixed learning rate.

GA component: On a regular basis, a genetic algorithm will be applied to those classifiers in $[A]_t$. In particular, proportional to their fitness, two classifiers from $[A]_t$ will be randomly selected to produce offspring classifiers, which are further modified through the crossover and mutation operations.

3 Natural XCS Classifier System

Aimed at learning stochastic policies, based on XCS, a new NXCS classifier system will be developed in this section. We organize our discussion into three subsections. Subsection 3.1 introduces the concept of stochastic policy. The reinforcement component of NXCS is further presented in Subsection 3.2. Finally, Subsection 3.3 develops a policy parameter learning component.

3.1 Stochastic Policy

In comparison with XCS, each classifier cl in NXCS includes an additional *policy parameter*, denoted as θ_{cl}. At any time t, using all classifiers in the match set $[M]_t$, the probability of taking any action $a \in A$ is determined according to (2) below.

$$\pi_t(s_t, a) = \frac{\prod\limits_{cl \in [M]_t^a} e^{\theta_{cl}}}{\sum\limits_{b \in A} \left(\prod\limits_{cl \in [M]_t^b} e^{\theta_{cl}} \right)} \tag{2}$$

where $\pi_t(s, a)$ refers to a *stochastic policy* that assigns a certain probability for performing any action a in state s_t at time t.

We construct a policy parameter vector $\boldsymbol{\theta}_t$ to include θ_{cl} of every classifier cl belonging to the match set $[M]_t$, assuming a pre-defined global order on these classifiers. The policy $\pi_t(s, a)$ is subsequently viewed as a function of $\boldsymbol{\theta}_t$.

3.2 Prediction Reinforcement

NXCS follows a similar learning procedure as XCS. During a single learning step at time t, based on the match set $[M]_t$, an action will be selected according to its probability defined in (2). The chosen action is then performed. As a result, the environment transits to a new state s_{t+1} and a scalar reward r_{t+1} is observed. r_{t+1} is then applied to update the prediction of each classifier $cl \in [A]$ using the updating rule below.

$$p_{cl}(t+1) \leftarrow p_{cl}(t) + \beta \cdot \left(r_{t+1} + \gamma \sum_{a \in A} \pi_t(s_{t+1}, a) \cdot P_{t+1}(a) - p_{cl}(t) \right) \tag{3}$$

The prediction updating in NXCS is different from that of XCS as shown in (1). This is because every policy in NXCS is stochastic. Hence prediction cannot be updated by assuming that, at time $t+1$, the action that gives the highest prediction will always be performed.

3.3 Learning Policy Parameters

In this subsection, a policy gradient framework is adopted to learn policy parameters. In particular, because the learning performance J (i.e. the discounted accumulated reward in the long run) can be treated as a function of $\boldsymbol{\theta}$, a straightforward approach is to learn $\boldsymbol{\theta}$ based on

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \lambda \cdot \nabla_{\boldsymbol{\theta}_t} J \tag{4}$$

where λ is a fixed learning rate. Practical application often shows that learning through (4) can be slow and unstable [10]. Instead of using $\nabla_{\boldsymbol{\theta}_t} J$, a *natural gradient* concept proposed by Amari can be very helpful [1]. Theoretically, stochastic policies learned through NXCS are equivalent to a family of statistical models situated in a Riemannian parameter vector space of $\boldsymbol{\theta}$. Each point in the space corresponds to a specific stochastic policy. In such a *Riemannian space*, learning should be performed through the *natural gradient* of J, i.e. $\tilde{\nabla}_{\boldsymbol{\theta}_t} J$. Particularly, we have

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \lambda \cdot \tilde{\nabla}_{\boldsymbol{\theta}_t} J \tag{5}$$

where

$$\tilde{\nabla}_{\boldsymbol{\theta}_t} J = G(\boldsymbol{\theta}_t)^{-1} \cdot \nabla_{\boldsymbol{\theta}_t} J \tag{6}$$

$G(\boldsymbol{\theta}_t)^{-1}$ is the inverse matrix of $G(\boldsymbol{\theta}_t)$. $G(\boldsymbol{\theta}_t)$ stands for the *Fisher information matrix* [1] of the stochastic policy represented by $\boldsymbol{\theta}_t$. In line with (5) and (6), it can be shown eventually that

$$\tilde{\nabla}_{\boldsymbol{\theta}_t} J \propto \cdot \delta_t \cdot \nabla_{\boldsymbol{\theta}_t} \log \pi_t(s,a) \tag{7}$$

where

$$\delta_t = r_{t+1} + \gamma \cdot \sum_{a \in A} \pi_t(s_{t+1}, a) \cdot P_{t+1}(a) - \sum_{a \in A} \pi_t(s_t, a) \cdot P_t(a) \tag{8}$$

Based on (7), the updating rule for learning policy parameters is determined as

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \lambda \cdot \delta_t \cdot \boldsymbol{\psi}_{s,a} \tag{9}$$

The learning parameter λ in (9) will be set to the inverse of the maximum single-step reward in all experiments to be reported in Section 4. It can be verified that the computational complexity of the performance component in NXCS is $\mathrm{O}(|[M]_t|)$, which is the same as XCS. Meanwhile, the complexity of

the reinforcement component in NXCS is O $(Max(|[M]_t|, |[M]_{t+1}|))$. Whereas in XCS, the corresponding complexity is O $(Max(|[A]_t|, |[M]_{t+1}|))$, which should not appear significantly different in practice.

4 Experiment Results

Experiments on three reinforcement learning problems will be reported here. The Woods101 problem is a partially observable environment. It is used to understand whether NXCS can better cope with *perceptual aliasing* [7] than XCS and XCS$_\mu$. The Woods14 problem is further used to study the performance of NXCS on benchmark deterministic multi-step problems with long-delayed reward. Finally, we will study the reliability of the learning system on stochastic maze problems, specifically the Maze5ϵ problem.

4.1 Experiments on the Woods101 Problem

The Woods101 problem, as described in [9], is a small grid environment that consists of 10 empty positions and one *terminal state* F (i.e. the goal). The agent, in any state, may choose to perform one of eight alternative actions. In the absence of an obstacle, each alternative action will bring the agent to a different adjacent position. To allow continued learning, whenever the agent reaches the terminal state, it will receive a maximum reward of 1000 and will be immediately relocated to a new state selected uniformly at random from the 10 empty positions.

Because an agent can only observe its adjacent positions in the grid, two states in the Woods101 problem are indistinguishable. Whenever a deterministic policy is followed, the same action will be performed in both states. There is hence a chance for the agent to be trapped in a local optima [9]. In comparison, an agent can achieve better performance by learning stochastic policies.

Fig. 1 presents the performance of NXCS, XCS, and XCS$_\mu$ on the Woods101 problem. Also included in this figure is the performance of another LCS named RXCS. RXCS is a learning system recently proposed by us for learning stochastic policies without using the natural gradient learning method. The maximum population size in our experiments is set to 300 classifiers. To reduce randomness, 30 independent tests have been conducted for each LCS. The average performance obtained is depicted in this figure. The same practice is also applied to build other result figures included in this paper.

As can be seen from Fig. 1, NXCS appears to perform better than XCS and XCS$_\mu$ throughout the whole learning process. In particular, at the end of the experiment (i.e. 8000 learning problems), the average performances achieved by NXCS, XCS, and XCS$_\mu$ are 4.39, 63.09, and 40.35 respectively. By using two-tailed t-test, it can be confirmed that the performance of NXCS is statistically better than that of XCS and XCS$_\mu$. Specifically, the p-value is 1.5483×10^{-127} for the t-test between NXCS and XCS and it equals to 9.1783×10^{-146} for the t-test between NXCS and XCS$_\mu$. Both the two p-values are far less than 0.05,

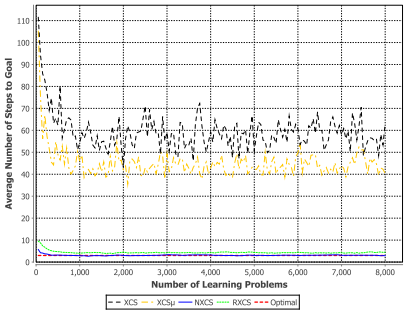

Fig. 1. Learning performance of NXCS, RXCS, XCS, and XCS$_\mu$ on the Woods101 problem. The performance is measured as the average number of actions to be performed by an agent in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

which is commonly used as the standard statistical significance level for t-tests. Meanwhile, we found that, after about 2000 learning problems, NXCS achieved an average performance that is very close to the theoretical optimum of 4.3 (with a small difference less than 0.1).

4.2 Experiments on the Woods14 Problem

In this Subsection, the performance of NXCS is further tested on the Woods14 problem. The Woods14 is a difficult benchmark reinforcement learning problem [3]. In particular, due to the problem's long-delayed reward, XCS has been reported as failing to solve the problem properly [3].

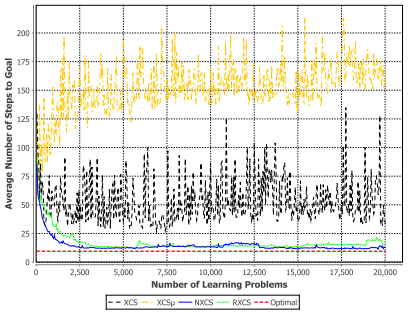

Fig. 2. Learning performance of NXCS, RXCS, XCS, and XCS$_\mu$ on the Woods14 problem. Performance is measured as the average number of actions an agent performs in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

Fig.2 depicts the learning performance of NXCS, XCS and XCS$_\mu$ on the Woods14 problem. Evidently, NXCS successfully stabilized its performance at an average of 12.619 after 3000 learning problems. In comparison, both XCS and XCS$_\mu$ cannot solve the problem properly, eventually achieving averages of 33.684 and 162.088 in performance respectively. The best policy in theory can achieve an average performance of 9.5 on the Woods14 problem. The performance of NXCS appears to be quite close to this theoretical optimum.

4.3 Experiments on Stochastic Maze Problems

In this subsection, we investigate the reliability of the learning systems in stochastic environments. Particularly, we have tested NXCS, XCS, and XCS$_\mu$ on several stochastic maze problems, including the Maze4ϵ, Maze5ϵ and Maze6ϵ problems. All our results consistently show that NXCS is more effective at handling environmental randomness. Specifically, to support this claim, we present here the experiment results on the Maze5ϵ problem, which is a stochastic extension of the benchmark Maze5 problem, as described in [9].

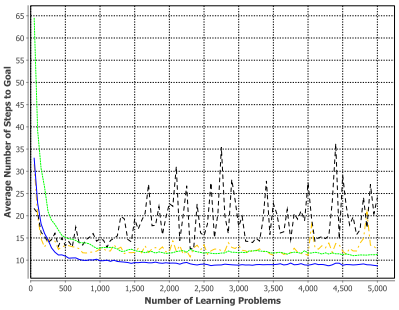

Fig. 3. Learning performance of NXCS, RXCS, XCS, and XCS$_\mu$ on the Maze5ϵ problem. Performance is measured as the average number of actions an agent performs in order to reach a goal.

As shown in Fig.3, NXCS has apparently performed better over the whole learning process, developing an average performance of 8.731 after 5000 learning problems. In line with the findings reported in [9], XCS failed to converge. Instead it exhibits large fluctuations and produces an average performance of 24.036 after 5000 learning problems. XCS$_\mu$ is more effective than XCS, achieving an average performances of 12.313 at the end of the experiment. If we perform a t-test between NXCS and XCS$_\mu$, a p-value of 4.758×10^{-32} is obtained, confirming that the performance difference between the two learning systems is statistically significant.

5 Conclusions

Based on XCS, this paper successfully developed a new natural XCS (i.e. NXCS) classifier system. Our research was inspired by the natural gradient learning technology. To the best of our knowledge, this paper presented the first study of natural gradient learning in XCS. Our method is general and can potentially be applied to many other LCSs. Meanwhile, our experiments showed that NXCS performed competitively with XCS and XCS_{μ}.

Looking into the future, we would hope to see interesting applications of NXCS to real-world problems that require sequential and stochastic decision making. The potential usefulness of NXCS for a wide range of machine learning tasks, including data mining problems, may also deserve in-depth investigation.

References

1. Amari, S.: Natural gradient works efficiently in learning. Neural Computation 10(2), 251–276 (1998)
2. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Natural actor-critic algorithms. Journal Automatica 45(11), 2471–2482 (2009)
3. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Gradient descent methods in learning classifier systems: improving xcs performance in multistep problems. IEEE Transactions on Evolutionary Computation (2005)
4. Butz, M.V., Wilson, S.W.: An Algorithmic Description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 253–272. Springer, Heidelberg (2002)
5. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
6. Holland, J.H.: Adaptation. In: Progress in Theoretical Biology, vol. 4, pp. 263–293. Academic Press (1976)
7. Lanzi, P.L.: An analysis of the memory mechanism of xcsm. In: Proceedings of the Third Genetic Programming Conference, pp. 643–651 (1998)
8. Lanzi, P.L.: Learning classifier systems: then and now. Evolutionary Intelligence (2008)
9. Lanzi, P.L., Colombetti, M.: An extension to the xcs classifier system for stochastic environments. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 353–360 (2000)
10. Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing, 1180–1190 (2008)
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
12. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems 12 (NIPS 1999), vol. 12, pp. 1057–1063. MIT Press (2000)
13. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation 3(2), 149–175 (1995)