Connectionist Classifier System Based on Accuracy in Autonomous Agent Control

A. S. Vasilyev
Decision Support Systems Group, Riga Technical University, 1/4 Meza street, Riga,
LV-1048, Latvia
E-mail: servent@apollo.lv

Abstract

In this paper a new connectionist classifier system based on accuracy is proposed, which uses a layer of competitive artificial neurons for decision-making. New algorithms of this system learning in multi-step problems are developed, where the correct result becomes known only at a certain system's step. The connectionist system is employed to control an autonomous agent in discrete environments.

1 Introduction

Impetuous development of artificial neural networks makes it possible to transfer many ideas from this area into adjacent areas. Further we will try to investigate an opportunity of mapping learning classifier systems (LCS) [1] into artificial neural networks (ANN) and to propose learning algorithms for a hybrid connectionist classifier system (CLCS) [2] intended to solve multi-step problems.

An LCS is a self-learning system based on syntactically simple rules (classifiers). Numerous classifiers can be activated at the same time, therefore the information is being processed in parallel. Each classifier has its strength that shows its current usefulness in the system. The strength varies as experience is accumulated.

2 Structure

The topology of a hybrid connectionist classifier system is based on the structure proposed in [3] assuming that binary messages $A_{msg} = \{0, 1\}$ in the LCS are replaced by bipolar messages $A_{msg} = \{-1, +1\}$, *i.e.* 0 corresponds to -1.

Due to this assumption, a LCS can be represented as a competitive two-layer artificial neural network. Each classifier is represented by a neuron in the hidden layer. Each bit of the input message is associated with an input element of the network. Output layer neurons are associated with the output effectors (actions).

Weights are bipolar (either +1 or -1). Connections between inputs and neurons of the hidden layer with weight +1 correspond to 1 in the classifier condition. Connections between inputs and neurons of the hidden layer with weight -1 correspond to 0 in A_{cond}. The absence of connections corresponds to # (don't care).

3 Learning

As it was marked above, CLCS consists of two layers of neurons: the hidden layer and the output layer. These layers are competitive and are updated by using reinforcement learning. There are two types of reinforcement signal in CLCS: $\forall r_1$ and $-1 < r_2 < 1$. The signal r_1 moves to the hidden layer and r_2 - to the output one.

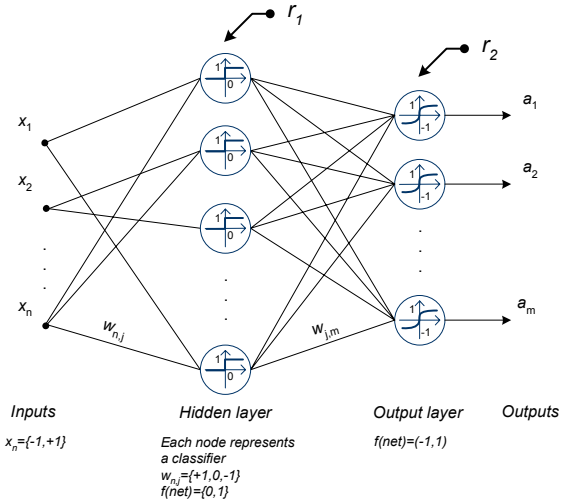

Fig. 1. Connectionist learning classifier system

One of the basic components of the CLCS output layer is the learning rate, *c*. Its adaptability improves a learning process. At the error increasing, the learning rate *c* increases by a pre-set factor γ_{incr}, *i.e.* $\Delta c = \gamma_{incr} c$. At the error decreasing, *c* decreases by a pre-set factor γ_{decr}, *i.e.* $\Delta c = -\gamma_{decr} c$.

3.1 Hidden layer learning

In most cases artificial neural network learning is based on the learning with a teacher (supervised) and without a teacher (unsupervised). In turn, LCS uses reinforcement learning approach.

Unlike ANN, where the weights are updated during learning, the learning of the CLCS hidden layer consists in updating classifiers' strengths. Only activated classifiers' strengths are updated. The hidden layer learning is carried out by any reinforcement learning algorithms, *i.e.* "bucket brigade" or Q-Learning.

3.2 Accumulation output layer learning

For the multi-step problem the reward *r* becomes known only at time *l*, therefore in this ease it is better to use the reinforcement type of learning, for example, a temporal-difference (TD)

learning. However TD learning is used for prediction of reward r and can be applied for a critic learning. However our task is to teach the controller.

For solving this problem we will describe a new approach for learning competitive ANNs. It uses the concepts of "winner takes all" and temporal difference learning.

The learning is based on the premise that one of the neurons m in the layer has the maximum response to input x. This neuron is declared the *winner*.

$$w_m^t x = \max_{i=1,2,\ldots,p} (w_i^t x) .$$

As a result of this winning event at step k, the weight vector appears

$$w_{m_k} = [w_{m1}, w_{m2}, \ldots, w_{mn}]_k^t .$$
$$out = f(w_m^t x) ,$$

where f is a continuously differentiable function.
The gradient is:

$$\nabla_w out = f'(w_m^t x) \cdot x .$$

Weights are updated for the winning neuron only, as follows:

$$w \leftarrow w + s \sum_{k=1}^{l} c\lambda^{(l-k)} \nabla_w out ,$$

where $s = \mathrm{sgn}(r)$ is the coefficient (either -1 or +1) which shows negative or positive reinforcement; c is the learning rate, $0 < c \le 1$; λ is the absolute value of the reinforcement signal r, $\lambda = |r|$, $0 < \lambda \le 1$. When $\lambda \to 0$, weights are not updated, *i.e.* $\Delta w \to 0$; for $\lambda = 1$ the weight changes at every step are identical.

At the beginning, all won neurons and input vectors x are stored. The weight updating occurs when the reward r at time l has been obtained. The closer the reward (the step l), the more updates in weights are.

The accumulation algorithm of the output layer learning looks as follows:

1. Initialize weight vectors, w_i, $i = 1,2,\ldots,p,$ with random values $\in (0,1)$

2. While not end do

(Main cycle)

2.1. For $k = 1,\ldots,l$ do

(Save each input vector x and index of won neuron m)

$x_k \leftarrow x$

$w_m^t x = \max_{i=1,2,\ldots,p} (w_i^t x)$

$m_k \leftarrow m$

2.2 For $i = 1,...,l$ do

(Update empty table t of weights when reinforcement r becomes known)

$\Delta t_{m_i} = sc\lambda^{(l-i)} f'\left(w_{m_i}^t x_i\right) \cdot x_i$

2.3 For $i = 1,\ldots,l$ do

(Update weights w using table t)

$\Delta w_{m_i} = t_{m_i}$

As there is data accumulation and changes occur only when the signal r is known, a large memory space for storing temporal variables is necessary. There are required $inputs \times l$ elements for storing all inputs x during a sequence l. This accumulation is possible only for rather short sequences. This disadvantage can be bypassed by using "bucket brigade" learning.

3.2 "Bucket Brigade" output layer learning

This type of learning does not make any accumulation. The winner gives a part of its weights to the previous won neuron. The last neuron in the sequence l receives a reinforcement signal from the environment $0 < r < 1$.

A neuron m in the output layer has the maximum output signal. This is declared the *winner*.

$$w_m^t x = \max_{i=1,2,\ldots,p} (w_i^t x) .$$

The weight vector of the won neuron m at current step k is

$$w_{m_k} = [w_{m1}, w_{m2}, \ldots, w_{mn}]_k^t .$$

The matrix **A** of dimension $p \times p$ consists of elements $a_{ij} = 0,\ i \neq j$ and $a_{ij} = x_i,\ i = j$.

The iterative learning rule reads:

$$d = cw_{m_k} \times \mathrm{A}_k ,$$

$$\Delta w_{m_k} = -d\,,$$
$$\Delta w_{m_{k-1}} = \|d\| x_k + w_{m_{k-1}}\,,$$

where c is the learning rate, $0 < c \le 1$;
d is the vector of changes of the won neuron;
$\|d\|$ is the norm of vector (scalar).

Weights' change of the last winner (activated) at step l is:

$$\Delta w_{m_l} = r x_l\,.$$

The "bucket brigade" algorithm of the output neural layer learning is as follows:

1. Initialize weight vectors, w_i, $i = 1,2,\ldots,p,$ with random values $\in (0,1)$
2. While not end do

   (Main cycle)

   2.1. For $k = 1,\ldots,l$ do

   (Update weights w of the previous m_{k-1} and current m_k won neurons)

   $w_{m_k}^t x_k = \max_{i=1,2,..,p} (w_i^t x)_k$

   $d = c w_{m_k} \times \mathrm{A}_k$

   $\Delta w_{m_k} = -d$

   $\Delta w_{m_{k-1}} = \|d\| x_k + w_{m_{k-1}}$

   2.2. (Now we have reinforcement signal $r,$ update last won neuron)

   $\Delta w_{m_l} = r x_l$

This learning is similar to the classical classifier systems' learning algorithm except that there are no taxes. In the case of success, not only the last activated neuron is encouraged, but also the previous activated neurons obtain some fraction of the reinforcement signal. Note that the learning factor c is similar to the bid auction factor c_{bid} in the classical "bucket brigade" algorithm.

4 Genetic algorithms and covering procedure

Since each classifier is connected to each action and has the parameters showing strengths of these connections (weights), it is necessary to change the weights of a classifier at its modification or replacement by a new one. For this purpose, we will make the following additions to the procedures of genetic algorithms (GA) and covering:

- *Selection* operator instead of weakly adapted classifier k chooses a more adapted one, i, from population **P**. All m weights of the new classifier are copied instead of old weights.

$$w_{i,j} = w_{k,j},\ \text{for}\ j = 1,2...m\ .$$

- *Crossover* operator crosses conditional parts of classifiers i and k. Weights of classifiers are their averaged weights.

$$w_{i,j} = w_{k,j} = \frac{w_{i,j} + w_{k,j}}{2},\ \text{for}\ j = 1,2...m\ .$$

- *Covering* procedure creates a new classifier, which matches the given input message instead of the weak one. Weights of the new classifier are initialized randomly:

$$w_{i,j} = rand,\ \text{for}\ j = 1,2...m\ .$$

5 Connectionist classifier system XCS

Currently there are various LCS types (*e.g.* XCS [4]). For example, XCS can be easily mapped into CXCS (Fig. 2).

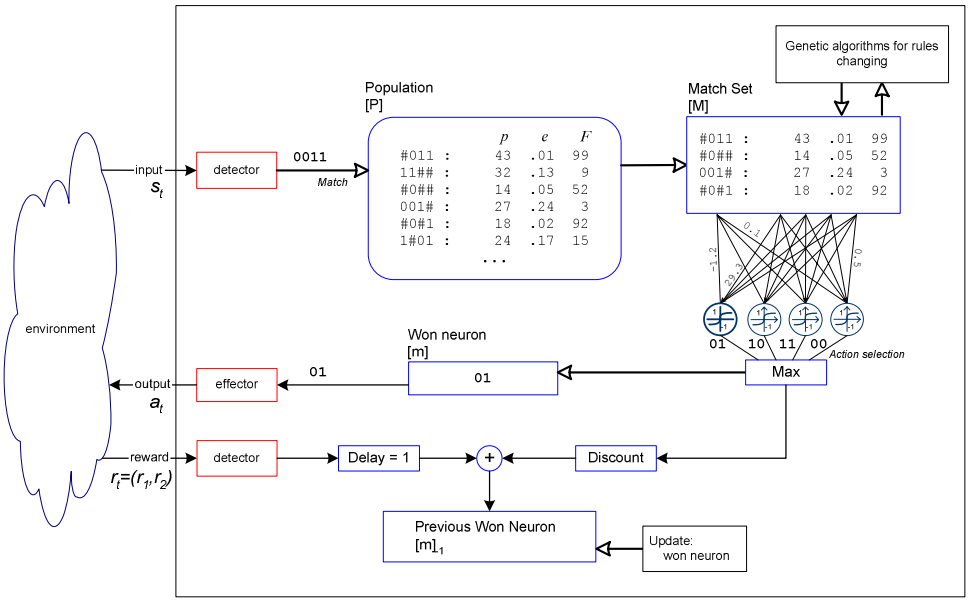

Fig. 2. Connectionist XCS

For XCS transformation into a connectionist classifier system it is necessary to substitute a set of activated competitive neurons for activation set **H**.

Consider the behaviour of an agent (Animat) which operates in the discrete environments Maze5 and Maze10 (Fig. 3), using a classifier system CXCS. The main objective of the agent is to collect food (**F**) by making the least possible number of steps for it (50 steps maximum). The agent can move in eight directions to the non-occupied positions. The agent has sensors by means of which it obtains the vectors of local perceptions. An agent's initial position every time is chosen randomly.

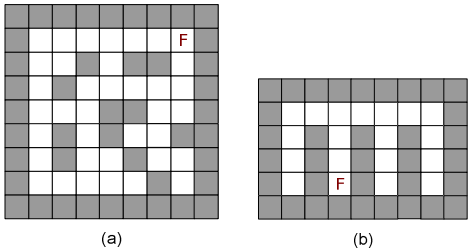

Fig. 3. Markov environment Maze5 (a) and non-Markov Maze10 (b)

All the graphs given below are averaged over 10 runs. The parameters used in ACS and XCS are default [4, 5]. Additional parameters for CXCS used in the experiments:

$(r_{pos})_2 = 10$ - a positive reward (food is found) for the hidden layer received from the environment;
$(r_{neg})_2 = -0.6$ - a negative reward (food is not found) for the hidden layer received from the environment;
$c = 0.016$ - hidden layer's learning rate;
$\gamma_{incr} = 1.05$ - increase factor of the learning rate c;
$\gamma_{decr} = 0.7$ - decrease factor of the learning rate c;
$c_{min} = 0.0001$ - minimum learning rate c;
$c_{max} = 0.9$ - maximum learning rate c.

Let us compare the performance of a connectionist system XCS (CXCS) (see Fig.2) with simple XCS and ACS in the Animat task.

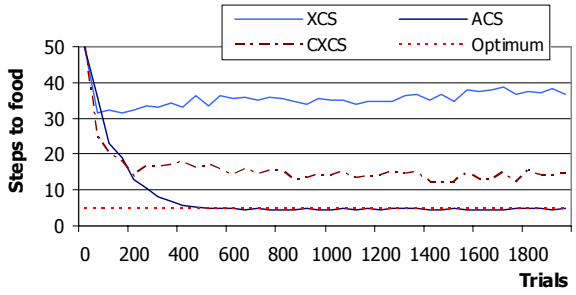

Fig. 4. Classifier systems' performance in Markov environment Maze5

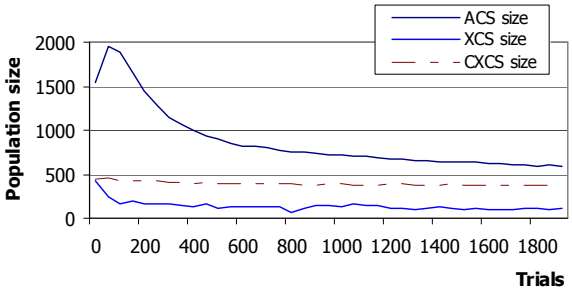

Fig. 5. Variation of classifiers' population size in Maze5

The CXCS shows much better results in Maze5 in comparing with the XCS. However, as well as XCS it cannot find an optimum. For example, at the 200-th trial (Fig. 4) a CXCS finds the goal object (food) within 12 steps on the average, while an XCS finds it in 32 steps. The best results are achieved by an ACS. The population size of ACS, XCS and CXCS classifier systems is time-variant (Fig. 5).

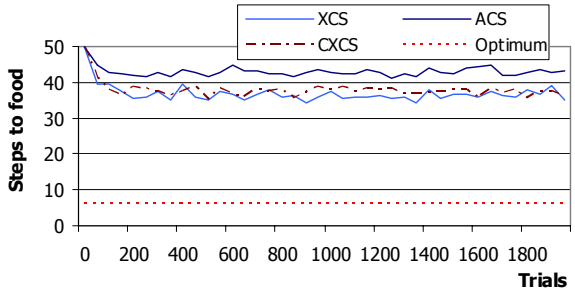

Fig. 6. Classifier systems' performance in non-Markov environment Maze10

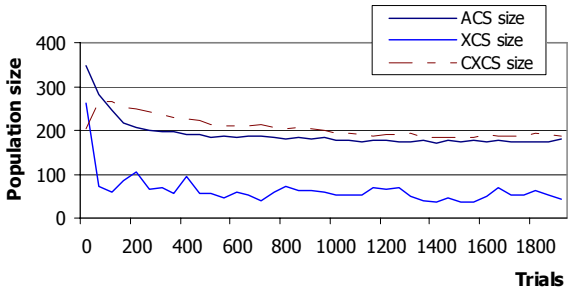

Fig. 7. Variation of classifier' population size in Maze10

Learning classifier systems used in the experiments do not possess memory [6] and action chunking procedure. That is why in non-Markov environment Maze10 (Fig. 6, Fig. 7) they cannot find an optimum, since for navigation to be efficient it is not enough to have only local perception information.

7 Conclusion

In this paper the possibility of using competitive neural networks in classifier systems was studied.

Two methods of CXCS learning have been developed that are based on reinforcement learning: accumulation and "bucket brigade" type. These competitive ANN learning methods can also be applied, independently of the CXCS, to the problems where the exact sequence of actions is not known, however it is known whether this sequence results in a positive reward or not.

The main CXCS advantage is that during the learning process not only classifiers' strengths are updated, but also actions are adjusted. In the case of unsuccessful actions the values of the parameters responsible for activation of these actions decrease. This increases the probability of choosing another action for the same situation. As a result, the number of classifiers necessary for CXCS is less than that for LCS. Apart from that, CXCS is less dependent on the genetic algorithm procedure.

A comparison of simple classifier systems and connectionist classifier systems was carried out in the agent navigation task (Animat) in Markov and non-Markov environments. However, in some cases the CXCS does not show an optimal performance.

References

[1] J. H. Holland, Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press, 1975.

[2] A. S. Vasilyev, Synergetic Approach in Adaptive Systems. Master's Thesis, Transport and Telecommunication Institute, Riga, Latvia, supervised by Prof. Borisov, A.N., 2002.

[3] R. E. Smith & H. B. Gribbs, Is a classifier system a type of neural network? *Evolutionary Computation*, 2(1), pp. 19-36, 1994.

[4] 4. Wilson, S.W. Classifier based on accuracy. *Evolution Computation*, 3(2), pp. 149-175, 1995.

[5] W. Stolzmann, An introduction to anticipatory classifier systems. P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS'99, LNAI 1813, pp. 175-194, 2000.

[6] 6. Lanzi, P. L. Adding Memory to XCS. In Proceedings of the *IEEE Conference on Evolutionary Computation* (ICES98), IEEE Press, 1998.