SEQUENTIAL AND REINFORCEMENT LEARNING FOR NEUROCONTROL

James J. Govindhasamy[†], Seán F. McLoone[††], George W. Irwin[†] and Vijanth.S. Asirvadam[†††]

[†]*Intelligent Systems and Control Research Group, Queen's University Belfast, Belfast BT9 5AH, N. Ireland, UK.*

[††]*Dept. of Electronic Engineering National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland.*

[†††]*Faculty of Information and Science Technology, Multimedia University, Malacca, Malaysia.*

E-mail: *j.govindasamy@ee.qub.ac.uk*

Abstract: This paper presents a novel sequential learning neural network implementation of action dependent adaptive critics. Sequential learning neural networks provide a systematic way of adding neurons in response to new data features as well as removing neurons which cease to contribute to the overall performance of the network. The convergence rate of the sequential learning method is enhanced by applying a modified Recursive Prediction Error algorithm to adjust network parameters. The new methodology, which provides a fully autonomous controller, is benchmarked against the conventional MLP neurocontroller on a highly nonlinear inverted pendulum system and shown to achieve superior performance.

Keywords: Radial basis function, sequential learning, neural networks, action dependent adaptive critics, reinforcement learning, minimal update.

I. INTRODUCTION

Implementing reinforcement learning (RL) using neural networks requires a great deal of engineering intuition in relation to determining the appropriate action and critic network structures and tuning the learning parameters. Thus, claims that RL systems are able to learn from scratch without *a priori* knowledge are not entirely true. In this contribution sequential learning neural networks are investigated as a means of obtaining a truly autonomous learning system.

Sequential learning neural networks (also known as 'constructive', 'incremental', or 'growing' networks) employ a learning procedure that involves growing and/or pruning networks iteratively as the training data is presented. Learning is achieved through a combination of new neuron allocation and parameter adjustment of existing neurons. New neurons are added if presented training patterns fall outside the range of existing network neurons. Otherwise the network parameters are adapted to better fit the patterns. This procedure is usually combined with pruning where neurons which contribute little to the overall network response over an extended period of time are removed. The seminal paper by Platt (1991), proved that sequential learning using a constructive technique, called resource allocation networks (RAN) is suitable for online modelling. Since then there have been many publications on research into application of this concept to supervised learning problems, e.g.: Kadirkamanthan and Niranjan (1993), Molina and Niranjan (1996) and Yingwei *et al*., (1997).

To date there have only been a handful of publications which explore the use of sequential learning neural networks with RL algorithms. In most of these implementations (e.g. Shiraga *et al.,* 2002 and Rivest and Precup, 2003) a sliding data window is used to achieve pseudo on-line learning.

This paper presents a novel sequential learning model-free action dependent adaptive critic (ADAC) for online control of a highly nonlinear process. This methodology overcomes the limitation of an *a priori* fixed network architecture normally associated with ADAC by extending the search to the entire weight space of the neural network topology. It also searches for a near minimal network size which suits the complexity of the learning task and thus increases the speed and efficiency of computation. This ultimately results in a fully autonomous controller.

The paper is organised as follows. Section 2 provides a brief description of the ADAC framework while the neural network implementation is discussed in Section 3. Section 4 describes the inverted pendulum case study, while Section 5 provides details of the

simulations performed and the results obtained. Conclusions and future work are given in section 6.

II. PRELIMINARIES

The fundamental solution to sequential optimisation or dynamic programming problems uses Bellman's Principle of Optimality (Bellman, 1957):... *an optimal trajectory has the property that no matter how the intermediate point is reached, the rest of the trajectory must coincide with an optimal trajectory as calculated with the intermediate point as the starting point*. This principle is applied by devising a "primary" reinforcement function or reward, $r(t)$, that incorporates a control objective for a particular scenario in one or more measurable variables. A secondary utility is then formed, which incorporates the desired control objective through time, the so-called Bellman equation, expressed as

$$J(t) = \sum_{k=0}^{\infty} \gamma^k r(t+k) \tag{1}$$

where γ is a discount factor ($0 < \gamma < 1$), which determines the importance of the present reward as oppose to future ones. The reinforcement, $r(t)$, takes a binary form with $r(t) = 0$ when the event is successful (objective is met) and $r(t) = -1$ when it fails (when the objective is not met). Hence, the purpose of dynamic programming is to choose a sequence of control actions to maximise $J(t)$, the cost-to-go. Unfortunately, this optimisation method is computationally untenable due to the complexity of the backward numerical solution process required, i.e. as a result of the "curse of dimensionality" for real problems. Thus, there is a need for more tractable approximation methods. The basis for such methods is a useful identity derived from Eq. 1, called the Bellman Recursion equation,

$$J(t) = r(t) + \gamma J(t+1) \tag{2}$$

Si and Wang (2001), formulated a modified version of Eq. 2, where, instead of approximating $J(t)$, they proposed that the Critic Network approximates $J(t+1)$. This is done by defining the future accumulated reward-to-go at time t, as

$$\hat{J}(t) = r(t+1) + \gamma r(t+2) + \ldots \tag{3}$$

and using the Critic Network to provide $\hat{J}(t)$ as an approximation to $J(t+1)$ as illustrated in Figure 1. In this framework, the Critic Network can be trained by storing the cost at t - 1 i.e. $\hat{J}(t-1)$ and using the current cost $\hat{J}(t)$, and the current reward, $r(t)$ i.e.

$$\hat{J}(t-1) = r(t) + \gamma \hat{J}(t) \tag{4}$$

to determine the error between two successive estimates of $\hat{J}$, referred to as the temporal difference error:

$$e_c(t) = \gamma \hat{J}(t) - [\hat{J}(t-1) - r(t)] \tag{5}$$

The cost function that the Critic Network is trained to minimise is then defined as

$$E_c(t) = \sum_t e_c(t)^2 \tag{6}$$

When $E_c(t) = 0$ for all t, Eq. 6 reduces to

$$\begin{aligned} \hat{J}(t-1) &= r(t) + \gamma \hat{J}(t) \\ &= r(t) + \gamma[r(t+1) + \gamma \hat{J}(t+1)] \\ &= \ldots \\ &= \sum_{k=t+1}^{\infty} \gamma^{k-t-1} r(k) \end{aligned} \tag{7}$$

where $\hat{J}(\infty) = 0$. Hence, it can be seen that by minimising Eq. 6, the Critic Network output becomes an estimate of the cost function $J(t+1)$ in Eq. 2, i.e. the value of the cost function at the next sample time.

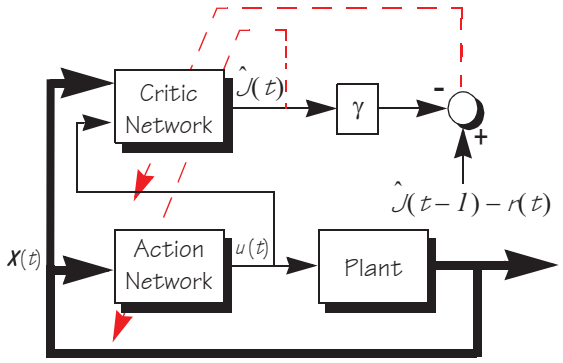

Figure 1 *Schematic of the Action Dependent Adaptive Critic scheme*

III. NEURAL NETWORKS IMPLEMENTATION

The sequential learning neural network architecture used here is based on the Radial Basis Function (RBF) network. The RBF network output is defined as:

$$y_k = \sum_{i=1}^{m} h_i \phi_i[\boldsymbol{x}_k(\boldsymbol{c}_i, \sigma_i)] \tag{8}$$

where h_i is the linear output weight that connects the i^{th} basis function, $\phi_i(.)$ to the output summer. The basis functions, $\phi_i(.)$ are usually localised Gaussian functions given by:

$$\phi[\boldsymbol{x}_k(\boldsymbol{c}_i, \sigma_i)] = exp\left(-\frac{\|\boldsymbol{x}_k - \boldsymbol{c}_i\|^2}{\sigma_i^2}\right) \tag{9}$$

where $\boldsymbol{c}_i$ and σ_i are the centre and width of the Gaussian function of the i^{th} hidden neuron. Practically RBFs show superior generalisation for on-line learning compared to multilayer perceptrons (MLP) due to the local nature of their neurons. This allows them to learn information at one operating point of a nonlinear process without degrading information learned at other operating regimes (McLoone, 2000). In other words training of RBF networks is not so susceptible to learning interference.

In the ADAC, the Critic Network is used to provide the approximation $\hat{J}(t)$. Starting with the prediction error defined in Eq. 5 and the instantaneous estimate of the Critic Network cost function is given by

$$E_c(t) = \frac{1}{2} e_c^{\,2}(t) \tag{10}$$

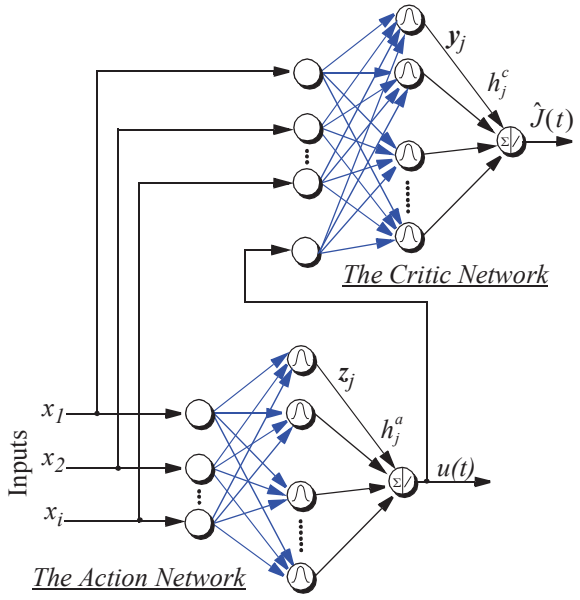

Figure 2 *The Action and Critic Network in an ADAC implementation*

The output of the Critic Network, shown in Figure 2, above is calculated in a feedforward manner and is expressed as follows

$$\hat{J}(t) = \sum_{j=1}^{N_h^c} h_j^c y_j(t) \tag{11}$$

where

$$y_j[\boldsymbol{c}_{cj}, \sigma_{cj}, \boldsymbol{x}_i] = exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{c}_{cj}\|^2}{\sigma_{cj}^{\,2}}\right) \tag{12}$$

Here, h_j^c, $\boldsymbol{c}_{cj}$ and σ_{cj} are the heights, centres and widths respectively, of the Gaussian basis function with output $\boldsymbol{y}_j$. N_h^c is the number of hidden neurons and $\boldsymbol{x}$ is the input vector. Note that the index N_i+1 includes $u(t)$ (i.e. $x_{N_i+i} = u(t)$), the output of the Action Network as shown in Figure 2.

The Action Network update is based on the prediction error given as

$$e_a(t) = \hat{J}(t) \tag{13}$$

and the objective function to be minimised is

$$E_a(t) = \frac{1}{2} e_a^{\,2}(t) \tag{14}$$

The output of the Action Network shown in Figure 2, is also calculated in a feedforward manner and is expressed as

$$u(t) = \sum_{j=1}^{N_h^a} h_j^a z_j(t) \tag{15}$$

where

$$z_j[\boldsymbol{c}_{aj}, \sigma_{aj}, \boldsymbol{x}_i] = exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{c}_{aj}\|^2}{\sigma_{aj}^{\,2}}\right) \tag{16}$$

Here, h_j^a, $\boldsymbol{c}_{aj}$ and σ_{aj} are the heights, centres and widths respectively, for the Gaussian basis function with output z_j in the action network and N_h^a is the number of hidden neurons and $\boldsymbol{x}$ is the input vector.

The weights-update rule for the Critic and Action Networks is based on a modified version of the recursive prediction error (RPE) algorithm (Chen *et al.*, 1990), expressed as follows,

$$P(t) = \frac{1}{\alpha}[P(t-1) - P(t-1)\ldots$$

$$\ldots(\nabla\psi[\boldsymbol{w}(t)])S^{-1}(t)(\nabla\psi^T[\boldsymbol{w}(t)])P(t-1)] \tag{17}$$

$$\text{with } P(t) = \frac{1}{trace[P(t)]}P(t) \tag{18}$$

$$\text{and } \nabla\psi[\boldsymbol{w}(t)] = \frac{\partial}{\partial \boldsymbol{w}} f[\boldsymbol{w}(t), \boldsymbol{x}(t)] \tag{19}$$

$$S(t) = \alpha + (\nabla\psi^T[\boldsymbol{w}(t)])P(t-1)(\nabla\psi[\boldsymbol{w}(t)]) \tag{20}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + P(t)(\nabla\psi[\boldsymbol{w}(t)])e(t) \tag{21}$$

where $P(t)$, is the inverse of the Gauss-Newton Hessian, which can be interpreted as the covariance matrix of the weight estimate $\boldsymbol{w}(t)$. Here, $\boldsymbol{w}(t)$ is a vector of all the parameters in either the Action or Critic networks, $\nabla\psi[\boldsymbol{w}(t)]$ is the corresponding gradient vector and $\alpha(t)$ is the forgetting factor which is normally set to a fixed value of $\alpha(t) < 1$.

Finally, $e(t)$ is the prediction error as defined in Eq. 5 for the Critic Network and Eq. 13 for the Action Network. The components of gradient vector for each network computed as follows

$$\begin{aligned}
\nabla\psi[h_j^c(t)] &= \frac{\partial E_c(t)}{\partial \hat{J}(t)} \cdot \frac{\partial \hat{J}(t)}{\partial y_j(t)} \cdot \frac{\partial y_j(t)}{\partial h_j^c(t)} \\
\nabla\psi[c_{cj}(t)] &= \frac{\partial E_c(t)}{\partial \hat{J}(t)} \cdot \frac{\partial \hat{J}(t)}{\partial y_j(t)} \cdot \frac{\partial y_j(t)}{\partial c_{cj}(t)} \\
\nabla\psi[\sigma_{cj}(t)] &= \frac{\partial E_c(t)}{\partial \hat{J}(t)} \cdot \frac{\partial \hat{J}(t)}{\partial y_j(t)} \cdot \frac{\partial y_j(t)}{\partial \sigma_{cj}(t)} \\
\nabla\psi[h_j^a(t)] &= \frac{\partial E_a(t)}{\partial \hat{J}(t)} \cdot \frac{\partial \hat{J}(t)}{\partial y_j(t)} \cdot \frac{\partial y_j(t)}{\partial u(t)} \cdot \frac{\partial u(t)}{\partial h_j^a(t)} \\
\nabla\psi[c_{aj}(t)] &= \frac{\partial E_a(t)}{\partial \hat{J}(t)} \cdot \frac{\partial \hat{J}(t)}{\partial y_j(t)} \cdot \frac{\partial y_j(t)}{\partial u(t)} \cdot \frac{\partial u(t)}{\partial c_{aj}(t)} \\
\nabla\psi[\sigma_{aj}(t)] &= \frac{\partial E_a(t)}{\partial \hat{J}(t)} \cdot \frac{\partial \hat{J}(t)}{\partial y_j(t)} \cdot \frac{\partial y_j(t)}{\partial u(t)} \cdot \frac{\partial u(t)}{\partial \sigma_{aj}(t)}
\end{aligned} \tag{22}$$

The sequential learning growth criterion considered here is an extension of the On-line Adaptive Centre Allocation (OLACA) algorithm (McLoone, 2000) which can explained as follows:

Let $(\boldsymbol{x}_t, y_t)$ be a new data point to be fitted by the RBF network. The growth criterion used is given by

$$d_{nc} = \min_i \|\boldsymbol{x}_t - \boldsymbol{c}_i\| > 2\alpha\sigma_{nc} \quad i = 1, 2, \ldots m \tag{23}$$

where d_{nc} is the distance between the current input vector, $\boldsymbol{x}_t$, and the centre of the nearest hidden neuron, $\boldsymbol{c}_i$, while σ_{nc} is the width of the nearest neuron. The term $2\alpha\sigma_{nc}$ is the maximum neuron separation allowed while the scalar α determines the degree of overlap between neurons and is usually set equal to 1.0.

If the criteria in Eq. 23 is not satisfied then all the network parameters are adapted using the RPE training algorithm. However if the growth criteria in Eq. 23 is satisfied then a new Gaussian basis function hidden neuron is assigned (shown in Figure 3) as follows:

$$\boldsymbol{c}_{N+1} = \boldsymbol{x}_k, \; h_{N+1} = e_k, \quad \sigma_{N+1} = \beta\frac{d_{nc}}{2} \tag{24}$$

where e_k is the deviation from the desired goal. This is problem dependent for the Action Network and is defined as Eq. 5 for the Critic Network. The scalar β, is a user-defined parameter (usually unity) which determines the degree of overlap between neurons.

The dimension of *P(t)*, is also increased accordingly, i.e.

$$P(t) = \begin{bmatrix} P(t-1) & O^T \\ O & I_{n_w} \end{bmatrix} \tag{25}$$

where the dimension n_w is equal to the number of new parameters associated with the new Gaussian basis function and O is an appropriately dimensioned null matrix.

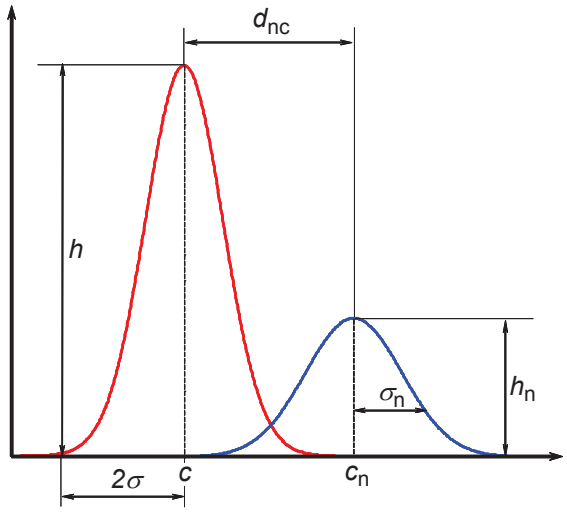

Figure 3 *The OLACA scheme*

The pruning procedure, which is based on YingWei *et al.* (1997), involves eliminating the Gaussian kernels (*GK*) that show the least contribution to the model output for the past *M* sample instants, and can be summarised as follows;

- Compute the output of all the Gaussian kernel functions, $i = 1, 2, \ldots, m$.

$$GK(t)_i = exp\left(-\frac{\|\boldsymbol{x}(t) - \boldsymbol{c}_i\|^2}{\sigma_i^2}\right) \tag{26}$$

- Find the largest absolute Gaussian basis function output value

$$GK_{max} = max(|GK(t)_i|) \text{ and } i = 1, 2, \ldots, m \tag{27}$$

- Determine the normalised contribution factor for each basis function:

$$\omega_i = \left|\frac{GK(t)_i}{GK_{max}}\right| \text{ and } i = 1, 2, \ldots, m \tag{28}$$

If $\omega_j < \delta$ ($\delta \ll 1$, normally set to 1 x 10^{-5}) for *M* consecutive sample instants, then prune the j^{th} hidden neuron and reduce the dimensionality of $\boldsymbol{w}(t)$, $\psi[\boldsymbol{w}(t)]$ and $P(t)$. The window size, *M,* is problem dependent and is normally set between 20 and 100.

IV. THE INVERTED PENDULUM

To test the performance of the proposed sequential learning neural networks, the ADAC was trained to control a highly nonlinear inverted pendulum system. This consisted of a pendulum hinged to the top of a wheeled cart that travels along a track, as shown in Figure 4. The goal was to apply a sequence of right and left forces of fixed magnitude to the cart such that the pole is balanced and the cart does not hit the end of the track, i.e. bang-bang control. A zero magnitude force was not permitted. The state of the cart-pole system had to be kept out of certain regions of the state space. There is no unique solution. Any state space trajectory that did not pass through the regions to be avoided was acceptable. The only information provided regarding the goal of the task was the reinforcement signal, $r(t)$, which signals a failure when either the pole fell outside $\pm 12^{o}$ or the cart hit the bounds of the track at $\pm$ 2.4 m.

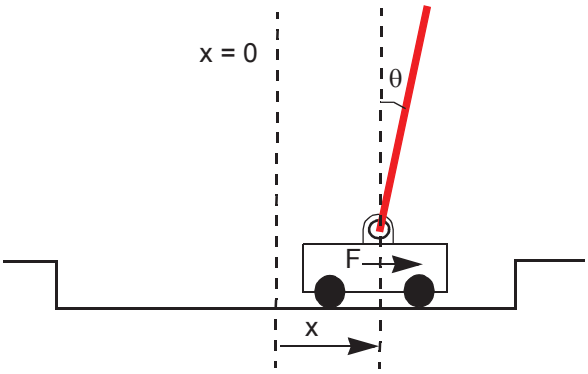

Figure 4 *The inverted pendulum system*

The simulation was based on a detailed analytical model of the system which included all the nonlinearities and reactive forces of the physical system such as friction (Si and Wang, 2001). The inverted pendulum was constrained to move within the vertical plane. The input state vector sample time at time t was specified by four real-valued variables, $x(t)$, $\dot{x}(t)$, $\theta(t)$, and $\dot{\theta}(t)$ defined as follows:

$x(t)$ = the horizontal position of the cart, relative to the track, in meters,

$\dot{x}(t)$ = the horizontal velocity of the cart, in meters/second,

$\theta(t)$ = the angle between the pendulum and vertical, clockwise being positive, in degrees,

$\dot{\theta}(t)$ = the angular velocity of the pendulum, in degrees/second.

All the input states were normalised to lie within [-1,1] as required by the neural networks. The description of the dynamics of the inverted pendulum can be found in Si and Wang (2001).

V. SIMULATION AND RESULTS

The sequential learning ADAC was compared against Si and Wang's (2001) multilayer perceptron (MLP) neural network implementation to gauge its performance. The simulation studies were based on 100 runs, each run consisting of 10 trials where the controller had to control the inverted pendulum within set boundaries. The external reinforcement signal was defined as

$$r(t) = \begin{cases} -1, \text{ If } |\theta| > 12° \text{ or } |x| > 2.4m \\ 0, \text{ otherwise} \end{cases} \qquad (29)$$

The controller was considered successful if it managed to balance the inverted pendulum for 600,000 time steps, where each time step was 0.02 seconds (i.e. for 3 hours and 20 mins). If after 10 trials the controller still failed to control the pendulum, that run was considered a failure and a new run was initiated with the pendulum states set to zero and all the networks weights initialised randomly. The nominal Action Network size in Si and Wang's (2001) MLP configuration for this same application was a 4-6-1. The maximum number of neurons was set at 10 for both RBF networks. The height of the new Gaussian function in the sequential learning methodology, as defined by Eq. 24, was given as the larger of the following deviations

$$h_{N+1} = \Delta\theta(t) \text{ or } h_{N+1} = \Delta x(t) \qquad (30)$$

where $\Delta\theta(t)$ and $\Delta x(t)$ are the normalised deviations of the angle of the pendulum from the vertical position (i.e. at $0°$) and the cart position from the middle of the track respectively.

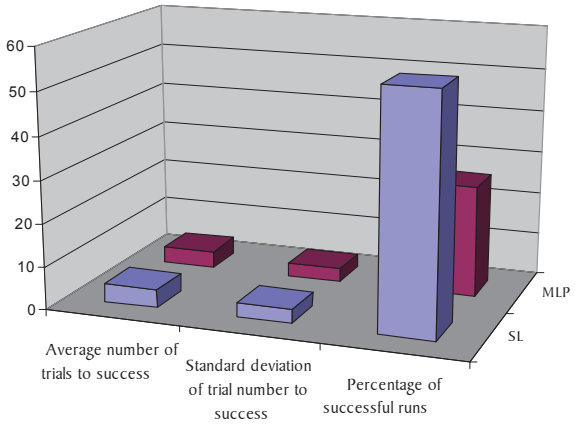

Figure 5 *Performance comparison between the sequential learning ADAC and the MLP structure based on 100 runs*

Figure 5 shows that the sequential learning ADAC was able to achieve double the success rate of the fixed structure MLP implementation. The superior performance of the ADAC controller compared to the

MLP structure may be explained by the ability of the sequential ADAC topology to adapt more rapidly to new data by adding new neurons (as seen in Figure 6). It is also aided by the robustness of the RBF topology to learning interfernce.

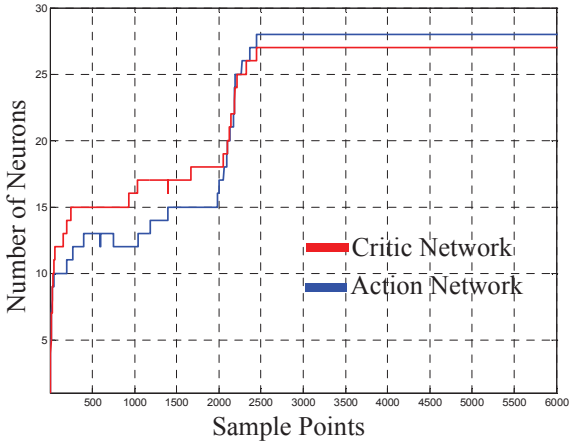

Figure 6 *The growth profiles of the ADAC networks*

Topology	Mean of $\hat{J}$	Mean of e_c	Comp. Time (secs)
MLP	1.009×10^{-5}	38.999×10^{-4}	160.86
ADAC	0.264×10^{-5}	0.001×10^{-4}	924.00

Table 1: Algorithm efficiency comparison based on typical single successful run

A performance comparison based on a typical successful run by both the sequential learning method and standard MLP networks is given in Table 1 above. The higher computation time is expected due to the learning algorithm chosen, compared to the simple backpropagation training algorithm that was used by Si and Wang (2001). However, for the approximation performance, i.e. the mean prediction error, E[e_c], the sequential learning ADAC was found to be superior to the conventional method.

VI. CONCLUSIONS AND FUTURE WORK

A new sequential learning ADAC methodology has been proposed which grows its structure to suit the complexity of the learning task. This provides an effective way of obtaining a compact solution without *a priori* knowledge of the problem. In contrast, the conventional ADAC learning strategy proposed by Si and Wang (2001) requires systematic evaluation of several different networks to determine the optimum network sizes. Simulation results also indicate that the sequential learning ADAC, which can be regarded as a fully autonomous controller, out performs the fixed topology ADAC. Future work will look at implementing this technique on real applications such as laboratory-scale process control applications.

ACKNOWLEDGEMENT

The first author wishes to acknowledge the financial support of Seagate Technology Media (Ireland) Ltd. and Queen's University Belfast.

REFERENCES

Anderson, C. W., "Q-Learning with Hidden-Unit Restarting", *Advances in Neural Information Processing Systems*, 5, pp. 81-88, 1993.

Bellman, R. E., "Dynamic Programming", Princeton, NJ, Princeton University Press, 1957.

Chen S., Billings S.A and Grant P.M., "Non-linear system Identification using Neural Networks", International Journal of Control, vol. 51, no. 6, pp. 1191-1214, 1990.

Kadirkamanathan, V. and Niranjan, M., "A function estimation approach to sequential learning with neural networks", *Neural Computation*, Vol. 5, no. 6, pp. 954-975, 1993.

McLoone S.F., "Neural Network Identification of AVR Loop Dynamics: A Comparison Between MLPs and RBFs.", CD-ROM *Pre-Print 35th Universities' Power Engineering Conference* (UPEC 2000), Belfast, U.K., Sept. 6-8, 2000.

Molina, C., and Niranjan, M., "Pruning with replacement on limited resource allocating networks by F-projections", *Neural Computation*, Vol. 8, no. 4, pp. 855-868, 1996.

Platt, J. C., "A resource-allocating network for function interpolation", *Neural Computation*, vol. 3, pp. 213-225, 1991.

Potocnik, P., and Grabec, I., "Adaptive self-tuning neurocontrol", *Mathematics and Computers in Simulation*, no. 51, pp. 201-207, 2000.

Rivest, F., and Precup, D., "Combining TD-Learning with Cascade-correlation Networks", *Proc. of the 20th Int. Conf. on Machine Learning* (IMCL-2003), pp. 632-639, 2003.

Shiraga, N., Ozawa, S., and Shigeo, A., "A Reinforcement Learning Algorithm for Neural Networks with Incremental Learning Ability", Proc. of the Int. Conf. on Neural Information Processing 2002 (ICONIPS' 2002), Vol. 5, pp. 2566 - 2570, 2002.

Si, J., and Wang, Y. T., "Online Learning Control by Association and Reinforcement", *IEEE Transactions on Neural networks,* Vol. 12, No. 2, pp. 264-276, 2001.

Sutton R. S., "Learning to Predict by the Method of Temporal Differences", *Machine Learning*, Vol. 3, pp. 9-44, 1988.

Yingwei L., Sundararajan N. and Saratchandran P., "Identification of Time-Varying Nonlinear Systems Using Minimal Radial Basis Function Neural Networks.", *IEE Proceedings Control Theory Application*, vol. 144, no. 2, pp. 202-208, 1997.