Reinforcement Learning for Online Control and Optimisation

James J. Govindhasamy†, Seán F. McLoone††, George W. Irwin†
John J. French†††, Richard P. Doyle†††

†*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.*

†††*Seagate Technology Media Ltd., 99 Dowland Road, Aghanloo Industrial Estate Limavady BT49 OHR, N. Ireland, UK.*

Abstract

An intelligent controller has the ability to analyse an unknown situation and to respond to it accordingly. Approximate dynamic programming, or reinforcement learning as it is more commonly known, in the form of Adaptive Critic Designs (ACDs), falls into this category [56]. ACDs offer an interesting alternative for adaptive control and optimisation of highly nonlinear industrial processes. In this chapter, the action dependent adaptive critic (ADAC) [47] is used and a suitable second-order training algorithm is presented to ensure fast convergence and stability. The performance of the training algorithm is first compared in simulation for the control of an inverted pendulum. The ADAC scheme is then applied to the control of an aluminium subtrate disk grinding process where the learning is based on actual industrial historical data. Results here indicate that the ADAC controller can control the unloading thickness variation of the process to achieve a 33% reduction in rejects.

1.0 Introduction

Current control methodologies can generally be divided into model-based and model-free. The first contains conventional controllers, the second so-called intelligent controllers [38,4,5]. Conventional control designs involve constructing dynamic models of the target system and the use of mathematical techniques to derive the required control law. Therefore, when a mathematical model is difficult to obtain, either due to complexity or to the numerous uncertainties inherent in the system, conventional techniques are less useful. Intelligent control may offer a useful alternative in this situation.

An intelligent system learns from experience. As such, intelligent systems are adaptive. However, adaptive systems are not necessarily intelligent. Key features of adaptive control systems are that they

- continuously identify the *current* state of the system
- compare the *current* performance to the desired one and decide whether a change is necessary to achieve the desired performance;
- modify or update the system parameters to drive the control system to an optimum performance.

These three principles, identification, decision, and modification, are inherent in any adaptive system. A learning system on the other hand is said to be intelligent because it improves its control strategy based on past experience or performance. In other words, an adaptive systems regards the *current* state as novel (i.e. localisation), whereas a learning system correlates experience gained at previous plant operating regions with the *current* state and modifies its behaviour accordingly [19] for a more long term effect.

There are many examples in industrial control where conventional automatic control systems (e.g. self tuning controllers) are not yet sufficiently advanced to cater for nonlinear dynamics across different operating regions or to predict the effects of current controller changes in the longer term. This is certainly true of very large, highly interconnected and complex systems. In these situations an intelligent approach for evaluating possible control alternatives, can be of value. One such framework, called the Adaptive Critic Design, was proposed by Werbos [55,56]. The design of nonlinear optimal neurocontrollers using this ACD paradigm, is currently attracting much renewed interest in the academic community.

However, closer examination of the current literature suggests that, a number of restrictive assumptions have had to be introduced which run counter to the original ADC concept. Wu [61,62], Chan [8], Zeng *et al.* [64] and Riedmiller [42] all assumed *a priori* knowledge of the plant in selecting the control action. Ernst *et al.* [12], Park [36,37], Venayagamoorty *et al.* [51,52,53], Iyer and Wunsch [25], Radhakant and Balakrishan [41], Sofge and White [46,57] trained their neurocontrollers offline using a plant model. While Hoskin and Himmelblau [21] successfully implemented an online model-free Adaptive Heuristic Critic (AHC) [1,2], the control employed was constrained to be of bang-bang form.

AHC and Q-learning, variants of the Adaptive Critic Designs, will be discussed in Chapter 10 in the context of multi-agent control within an internet environment. In this chapter, the model-free, action dependent adaptive critic (ADAC) design of Si and Wang [47] is extended to produce a fully online neurocontroller

without the necessity to store plant data during a successful run. In particular, the potential limitations of Si and Wang's stochastic backpropagation training approach, in terms of poor convergence and parameter shadowing [32] are avoided. This is done by introducing a second order training algorithm, based on the recursive Levenberg-Marquardt (RLM) [34,35] approach. This training algorithm incorporates the temporal difference (TD) strategy [48,49], and is hence designated TD-RLM.

The performance of our new ADAC scheme for reinforcement learning is validated using simulation results from an "inverted pendulum" (pole-balancing) control task. This is often used as an example of the inherently unstable, multiple-output, dynamic systems present in many balancing situations, like a two-legged walking robot or the aiming of a rocket thruster. It has also been widely used to demonstrate many modern and classical control engineering techniques.

Finally, results from a collaboration with Seagate Technology Ltd. on data from an actual industrial grinding process used in the manufacture of disk-drive platters suggest that the ADAC can achieve a 33% reduction in rejects compared to a proprietary controller, one of the first reported industrial applications of this emerging technology.

The chapter is organised as follows. Section 2 gives a detailed description of the ADAC framework and its neural network implementation. The training algorithm for the neural implementation is then compared in simulation on the control of an inverted pendulum in Section 3. Section 4 introduces the industrial grinding process case study and presents results on the application of ADACs to identification and control of the process. Conclusions appear in section 5.

2.0 Action Dependent Adaptive Critics

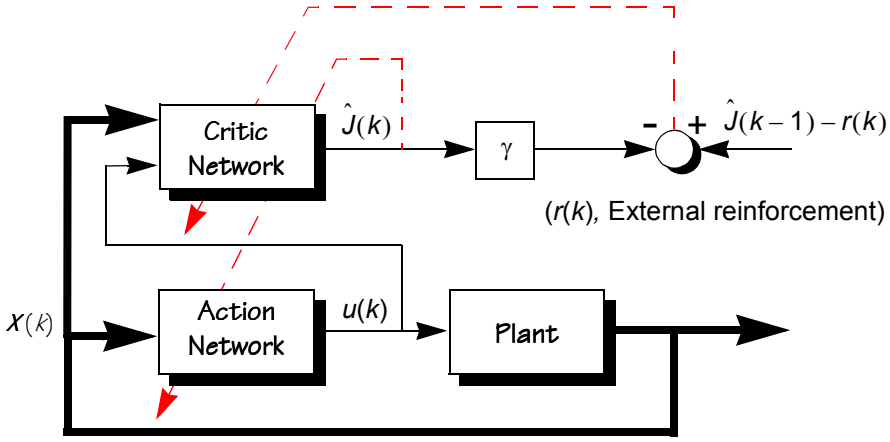

Figure 2-1 *Schematic of the Action Dependent Adaptive Critic scheme*

The algorithm used to illustrate the features of the Action Dependent Adaptive Critic (ADAC) shown in Figure 2-1, is based on Si and Wang [47] and belongs to the approximate dynamic programming family. Such methods were first introduced and formalised by Werbos [55][56]. It is useful to summarise how this method came to be used.

The fundamental solution to sequential optimisation problems relies on Bellman's Principle of Optimality [7]:... *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 in reinforcement learning by devising a "primary" reinforcement function or reward, $r(k)$, 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. This is called the Bellman Equation and is expressed as

$$J(k) = \sum_{i=0}^{\infty} \gamma^i r(k+i) \tag{2.1}$$

where γ is a discount factor ($0 < \gamma < 1$), which determines the importance of the present reward as opposed to future ones. The reinforcement, $r(k)$, is binary with $r(k) = 0$ when the event is successful (the objective is met) and $r(k) = -1$ when failure occurs (when the objective is not met). The purpose of dynamic programming is then to choose a sequence of control actions to maximise $J(k)$, which is also called the

cost-to-go. Unfortunately, such an optimisation is not feasible computationally due to the backward numerical solution process needed which requires future information for real problems. There is thus a need for a more tractable approximation method which uses the Bellman Recursion equation for the cost-to-go,

$$J(k) = r(k) + \gamma J(k+1) \tag{2.2}$$

Werbos proposed a variety of methods for estimating $J(k)$ using artificial neural networks as function approximators. These methods are called Adaptive Critics Designs (ACDs). This term is generally applied to any module that provides learning reinforcement to a second, Action Network module (i.e. a controller) [55].

The standard classification of these ACDs is based on the critic's inputs and outputs. In Heuristic Dynamic Programming (HDP) for example, the critic's output is an estimate of the value of $J(k)$ while in Dual Heuristic Programming (DHP) the critic's output is an estimate of the derivative of $J(k)$ with respect to the states. Globalised dual heuristic programming (GDHP), approximates both $J(k)$ and its derivatives by adaptation of the critic network. In the *action dependent* versions of HDP and DHP, the critic's inputs are augmented with the controller's output (action), hence the names ADHDP, ADDHP and ADGDHP. The reader is referred to [55,56] for more details of the characteristics and uses of these different methods. Only the ADHDP is discussed here as Si and Wang's [47] approach is closely related to this method.

For illustration, suppose a discrete nonlinear, time-varying system is defined as,

$$x(k+1) = f[x(k), u(k), k] \tag{2.3}$$

where $x \in \Re^n$ represents the state vector and $u \in \Re^m$ denotes the control action. The cost function is represented by

$$J[x(k)] = \sum_{i=k}^{\infty} \gamma^{i-k} r[x(i), u(i)] \tag{2.4}$$

where r is the reinforcement signal and γ is a discount factor ($0 < \gamma < 1$). The objective is to choose the control action, $u(i)$, $i=k, k+1, \ldots$, so that the cost J defined in Eq. 2.4 is minimised. However, in Eq. 2.2, the

future value of the cost $J(k+1)$ is required which is not known *a priori*. In the ACDs, an adaptive critic network is used to produce the required estimate of $J(k+1)$.

The adaptive critic network is trained to minimise the following error measured over time,

$$E_c(k) = \sum_k [\hat{J}(k) - r(k) - \gamma \hat{J}(k+1)]^2 \tag{2.5}$$

$\hat{J}(k)$, the output of the critic network, is given by

$$\hat{J}(k) = \hat{J}[x(k), u(k), \boldsymbol{W}_c] \tag{2.6}$$

where $\boldsymbol{W}_c$ are the parameters of the critic network. Here the reinforcement signal indicates the performance of the overall system, i.e. failure = -1, or success = 0. When $E_c(k) = 0$ for all k, Eq. 2.5 can be simplified to

$$\begin{aligned} \hat{J}(k) &= r(k) + \gamma \hat{J}(k+1) \\ &= r(k) + \gamma [r(k+1) + \gamma \hat{J}(k+2)] \\ &= \dots \\ &= \sum_{i=k}^{\infty} \gamma^{i-k} r(i) \end{aligned} \tag{2.7}$$

which is the same as Eq. 2.4. However, this approach to training would require a plant model to predict x(k+1) and consequently the cost-to-go, $\hat{J}(k+1)$, as shown in Figure 2-2 below.

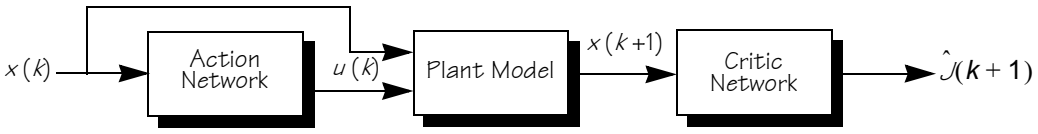

Figure 2-2 *Conventional ADHDP* [55,56]

At this point it is worth mentioning that, in adaptive critic designs, there are two partial derivative terms $\frac{\partial \hat{J}(k)}{\partial \boldsymbol{W}_c(k)}$ and $\frac{\partial \hat{J}(k+1)}{\partial \boldsymbol{W}_c(k)}$ in the backpropagation path from the Bellman equation. When adaptive critic designs were implemented without a model network, the second partial derivative term was simply ignored. This can be detrimental as demonstrated in [39], [40], and [54]. In previous implementations such as DHP and GDHP, a model network was employed to take into account the term $\frac{\partial \hat{J}(k+1)}{\partial \boldsymbol{W}_c(k)}$.

Si and Wang [47] proposed a method which resolved the dilemma of either ignoring the $\frac{\partial \hat{J}(k+1)}{\partial \boldsymbol{W}_c(k)}$ term, leading to poor learning accuracy, or including an additional system model network and hence more computation. These authors modified the Bellman Recursion in Eq. 2.2 so that, instead of approximating $J(k)$, the Critic Network would approximate $J(k+1)$. This is done by defining the future accumulated reward-to-go at time t, as

$$Q(k) = r(k+1) + \gamma r(k+2) + \ldots \tag{2.8}$$

and using the Critic Network to provide $Q(k)$ as an estimate of $J(k+1)$ i.e. $Q(k) = J(k+1)$, as shown in Figure 2-3 below.

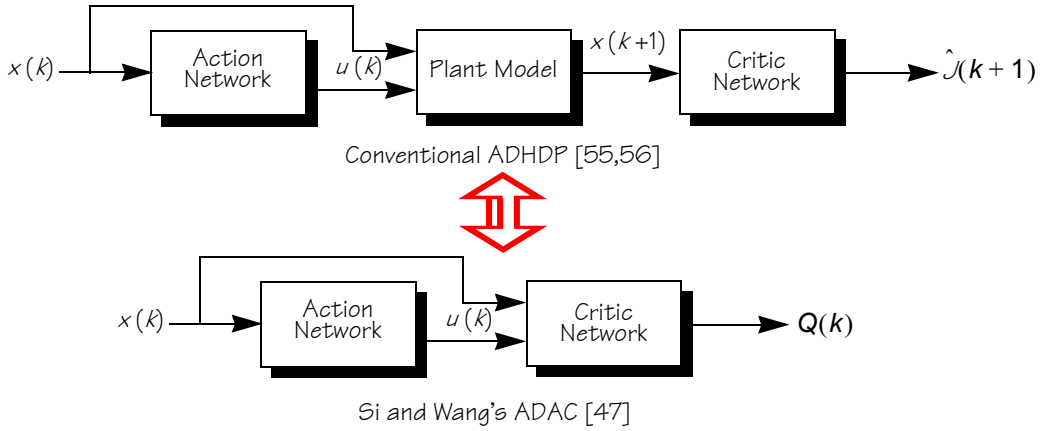

Figure 2-3 *Comparison of the conventional ADHDP and Si and Wang's ADAC*

In their method, the Critic Network is trained by storing the estimated cost at k-1, $Q(k-1)$. The current estimated cost $Q(k)$, and the current reward, $r(k)$, are then used to determine the temporal difference error (i.e. the error between two successive estimates of Q) as given by

$$e_c(t) = Q(k-1) - r(k) - \gamma Q(k) \tag{2.9}$$

where

$$Q(k-1) = r(k) + \gamma Q(k) \tag{2.10}$$

is the recursive form of Eq. 2.8. Thus the cost for the Critic Network to be minimised during training is defined as

$$E_c(k) = \sum_t [Q(k-1) - r(k) - \gamma Q(k)]^2 \tag{2.11}$$

When E_c (k) = 0 for all k, Eq. 2.11 simplifies to

$$\begin{aligned} Q(k-1) &= r(k) + \gamma Q(k) \\ &= r(k) + \gamma[r(k+1) + \gamma Q(k+1)] \\ &= \cdots \\ &= \sum_{i=k+1}^{\infty} \gamma^{i-k-1} r(i) \end{aligned} \tag{2.12}$$

where $Q(\infty) = 0$. Comparing Eqs. 2.7 and 2.12, it can be seen that by minimising $E_c(k)$, the Critic Network output then provides an estimate of $J(k+1)$ in Eq. 2.4, i.e. the value of the cost function in the immediate future.

Training can either be performed "backward-in-time" or "forward-in-time" [29]. In the case of backward-in-time training, the target output of the Critic Network at time t, is computed from the previous network output, $Q(k-1)$ using Eq. 2.10, that is:

$$Q(k) = \frac{1}{\gamma}[Q(k-1) - r(k)] \tag{2.13}$$

Thus the network is trained to realise the mapping

$$Critic \quad \{\boldsymbol{x}(k), u(k)\} \rightarrow \frac{1}{\gamma}[Q(k-1) - r(k)] \tag{2.14}$$

The arrangement is as shown in Figure 2-4.

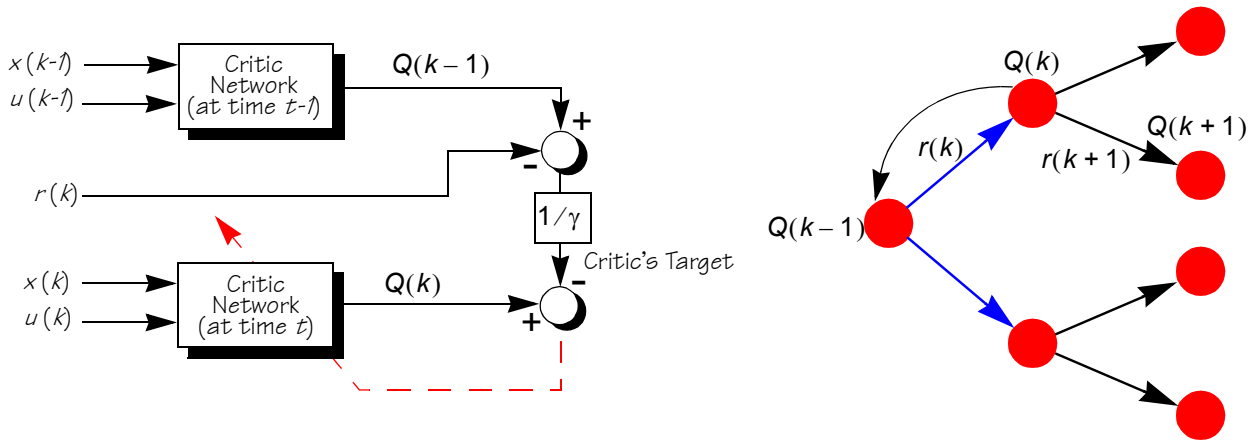

Figure 2-4 *Illustration of the "backward-in-time" mapping*

In the alternative forward-in-time approach at *k - 1*, the Critic Network is trained to produce the output $Q(k-1)$ and the output at time *t* is used to generate the required target output, that is: $r(k)+\gamma Q(k)$. Here the Critic Network realises the mapping

$$Critic\ \{\boldsymbol{x}(k-1), u(k-1)\} \rightarrow r(k)+\gamma Q(k) \tag{2.15}$$

and, $Q(k-1)$ is the network output. The forward in time arrangement is depicted in Figure 2-5.

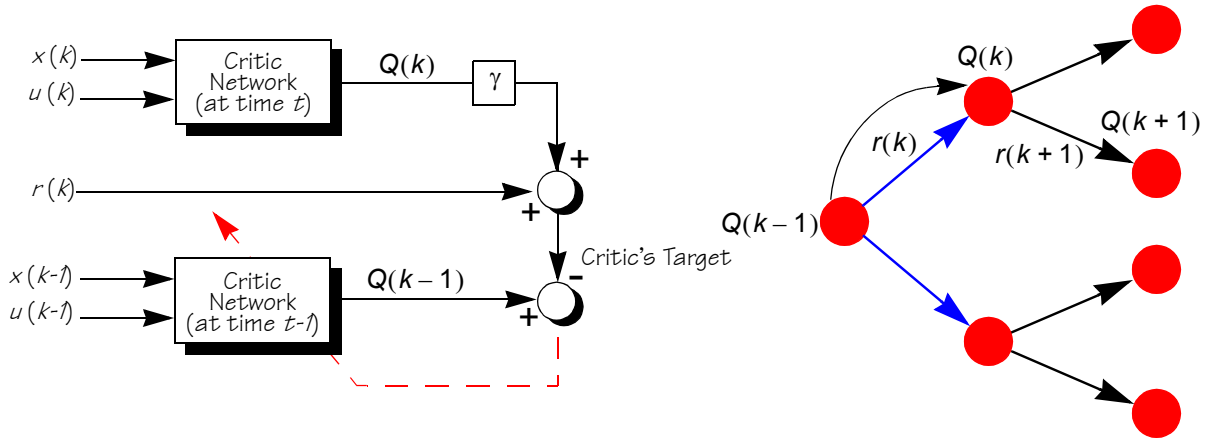

Figure 2-5 *Illustration of the "forward-in-time" mapping*

The Action Network is trained after the Critic Network, with the objective of maximising the critic output, $Q(k)$. This strategy indirectly enables the Action Network to produce favourable control actions and zero reward i.e. $r(k) = 0$. Thus, the target output from the Action Network for training purposes can be equated to zero, so that the Critic Network's output is as close to zero as possible. The mapping desired from the Action Network is given by

$$Action:\ \{x(k)\} \rightarrow \{Q(k) = 0\} \tag{2.16}$$

The training of the Action Network requires that it be connected to the Critic Network so that the target mapping in Eq. 2.16, then refers to the output of the *whole* network as shown in Figure 2-6.

Having defined the network mappings and cost functions, stochastic gradient based training algorithms can be used to train the networks. In Si and Wang's original ADAC design [47] networks were trained using a temporal difference error based stochastic backpropagation algorithm (TD-SBP). This is a first order

gradient descent method and therefore subject to poor convergence and parameter shadowing problems [32]. Here second-order training, in form of a recursive implementation of the Levenberg-Marquardt [34,35] algorithm (TD-RLM), is introduced to address these deficiencies and compared with Si and Wang's original TD-SBP approach. In each case the Critic and Action networks are implemented as Multilayer Perceptron neural networks.

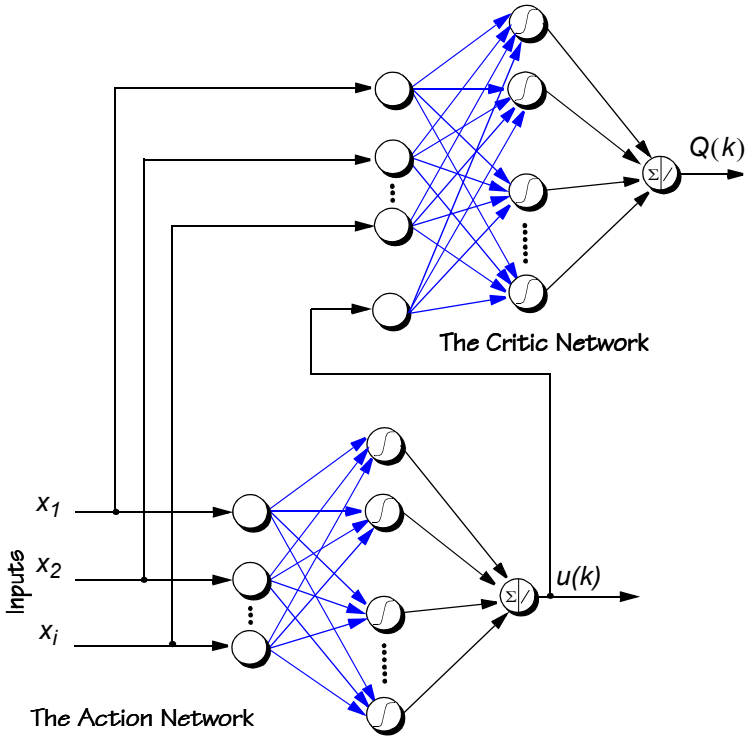

Figure 2-6 *The Action and Critic Network in an ADAC implementation*

2.1 First-Order Training Algorithm: Temporal Difference Stochastic Backpropagation (TD-SBP)

The output of the Critic Network is $Q(k)$ and its prediction error is given by

$$e_c(k) = \gamma Q(k) - [Q(k-1) - r(k)] \tag{2.17}$$

Thus, the objective function to be minimised is the instantaneous estimate of the mean-squared prediction error

$$E_c(k) = \frac{1}{2} e_c^{\,2}(k) \tag{2.18}$$

For a multilayer perceptron Critic Network (Figure 2-7), the output is calculated in a feedforward manner as follows:

$$g_j(k) = \sum_{i=1}^{N_i+1} w_{cij}^{NL}(k) x_i(k) \tag{2.19}$$

$$y_j(k) = \frac{1 - e^{-g_j(k)}}{1 + e^{-g_j(k)}} \tag{2.20}$$

where $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}_c^{NL}$ is the input to hidden layer (or nonlinear) weights, g is the input to the hidden layer nodes and y is the output of the hidden layer nodes. Note the index N_i+1 is to include $u(k)$ (i.e. $x_{N_i+i} = u(k)$), the output of the Action Network as shown in Figure 2-7. Finally, the output, $Q(k)$, is calculated as

$$Q(k) = \sum_{j=1}^{N_h} w_{cj}^{L}(k) y_j(k) \tag{2.21}$$

where $\boldsymbol{w}_c^{L}$ is the linear (or hidden to output layer) weights vector.

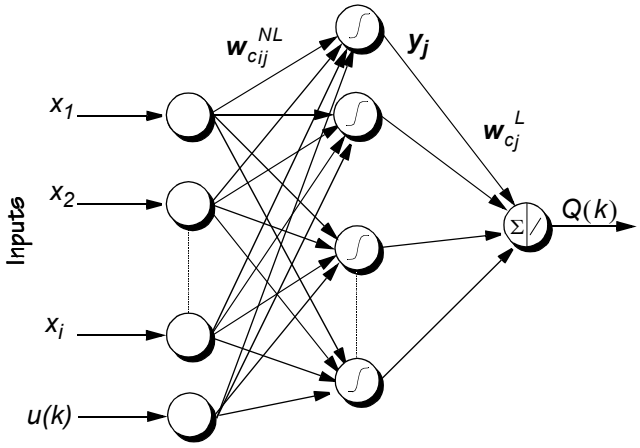

Figure 2-7 *Critic Network with the process states and control action, u(k) from the Action Network as inputs*

The weights-update rule for the Critic Network is based on a combination of gradient descent weight adaptation and temporal-difference (TD) learning [48,49]. The linear weights, $\boldsymbol{w}_c^{L}$, are updated as:

$$w_{cj}^{L}(k+1) = w_{cj}^{L}(k) + \Delta w_{cj}^{L}(k) \tag{2.22}$$

where

$$\begin{aligned}\Delta w_{cj}^{L}(k) &= -\beta_c(k)\left[\frac{\partial E_c(k)}{\partial w_{cj}^{L}(k)}\right] \\ &= -\beta_c(k)\left[\frac{\partial E_c(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial w_{cj}^{L}(k)}\right]\end{aligned} \tag{2.23}$$

The nonlinear weights, $\boldsymbol{w}_c^{NL}$, are correspondingly updated as:

$$w_{cij}^{NL}(k+1) = w_{cij}^{NL}(k) + \Delta w_{cij}^{NL}(k) \tag{2.24}$$

and,

$$\begin{aligned}\Delta w_{cij}^{NL}(k) &= -\beta_c(k)\left[\frac{\partial E_c(k)}{\partial w_{cij}^{NL}(k)}\right] \\ &= -\beta_c(k)\left[\frac{\partial E_c(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial y_j(k)} \cdot \frac{\partial y_j(k)}{\partial g_j(k)} \cdot \frac{\partial g_j(k)}{\partial w_{cij}^{NL}(k)}\right]\end{aligned} \tag{2.25}$$

In both cases, $\beta_c > 0$ is the learning rate which decreases with time.

The prediction error for the Action Network update is

$$e_a(k) = Q(k) \tag{2.26}$$

where the instantaneous estimate of the objective function to be minimised is given by

$$E_a(k) = \frac{1}{2} e_a^{\,2}(k) \tag{2.27}$$

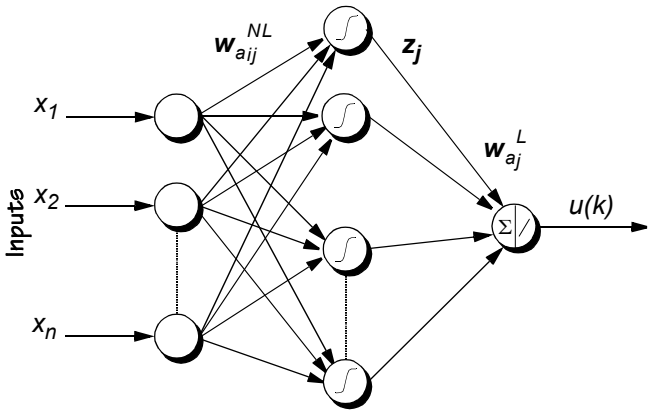

Figure 2-8 *Action Network topology receiving the process states*

The output of the Action Network shown in Figure 2-8, is calculated in a feedforward manner and is expressed as

$$f_j(k) = \sum_{i=1}^{N_i} w_{aij}^{NL}(k) x_i(k) \tag{2.28}$$

where $\boldsymbol{f}$ is the input to the hidden layer nodes, $\boldsymbol{w}_a^{NL}$ is the input to hidden layer weights, and $\boldsymbol{x}$ is the input vector. For the MLP architecture, the output is given by

$$u(k) = \sum_{j=1}^{N_h} w_{aj}^{L}(k) z_j(k) \tag{2.29}$$

where $\boldsymbol{w}_a^{L}$ is the linear weights vector and $\boldsymbol{z}$ is the output layer input function given as

$$z_j(k) = \frac{1 - e^{-f_j(k)}}{1 + e^{-f_j(k)}} \tag{2.30}$$

The weight-update rule for the Action Network is again based on gradient-descent adaptation. The linear weights, $\boldsymbol{w}_a^L$, adaptation rule is

$$w_{aj}^{L}(k+1) = w_{aj}^{L}(k) + \Delta w_{aj}^{L}(k) \tag{2.31}$$

where,

$$\begin{aligned} \Delta w_{aj}^{L}(k) &= -\beta_a(k)\left[\frac{\partial E_a(k)}{\partial w_{aj}^{L}(k)}\right] \\ &= -\beta_a(k)\left[\frac{\partial E_a(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial u(k)} \cdot \frac{\partial u(k)}{\partial w_{aj}^{L}(k)}\right] \end{aligned} \tag{2.32}$$

while the nonlinear weights, $\boldsymbol{w}_a^{NL}$ are adapted according to:

$$w_{aij}^{NL}(k+1) = w_{aij}^{NL}(k) + \Delta w_{aij}^{NL}(k) \tag{2.33}$$

and,

$$\begin{aligned} \Delta w_{aij}^{NL}(k) &= -\beta_a(k)\left[\frac{\partial E_c(k)}{\partial w_{aij}^{NL}(k)}\right] \\ &= -\beta_a(k)\left[\frac{\partial E_c(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial u(k)} \cdot \frac{\partial u(k)}{\partial z_j(k)} \cdot \frac{\partial z_j(k)}{\partial f_j(k)} \cdot \frac{\partial f_j(k)}{\partial w_{aij}^{NL}(k)}\right] \end{aligned} \tag{2.34}$$

Again $\beta_a > 0$ is a learning rate which decreases with time.

2.2 Second-order Training Algorithms

The reason for considering second order training methods, is that the original action dependent adaptive critic (ADAC) algorithm by Si *et al.* [47] used TD-SBP, which is known to have poor convergence properties [50]. In addition, it is susceptible to parameter shadowing when used on-line. This occurs when the network parameters continually adapt so that the network output tracks the desired output instead of converging to the desired model weights, $\boldsymbol{w}^*$ [32]. It is therefore logical to develop a more robust and stable training algorithm based on existing second order recursive schemes.

As with the SBP in the previous section, the second order recursive training schemes investigated here also seek to minimise a squared-error cost function, $E(k)$. Using the single data point available at the k^{th} sample instant, an instantaneous estimate of $E(k)$ is derived as

$$E(k) = \frac{1}{2}e^2(k) \tag{2.35}$$

where $e(k)$ is defined as

$$e_c(k) = \gamma Q(k) - [Q(k-1) - r(k)] \text{ and } e_a(k) = Q(k) \tag{2.36}$$

for the Critic and Action networks respectively. The use of the Levenberg-Marquardt algorithm here is motivated by the fact that it is often the best algorithm for offline training of neural networks. Hence the recursive version of this algorithm might be expected to yield similar performance benefits when training ADACs. There have been numerous studies demonstrating the superiority of second-order recursive estimation algorithms, such as recursive least squares (RLS), to first order algorithms, such as least mean squares (LMS), for linear-in-the-parameter models ([13, 26, 30, 63]). These algorithms can also be applied to the identification of nonlinear-in-the-parameter models as shown in [9], [10] and [11]. This is illustrated as follows. Assume that the predicted output at time k, for either of the Critic Network and Action Network, is represented by

$$\hat{y}(k) = a[\boldsymbol{w}(k), \boldsymbol{x}(k)] \tag{2.37}$$

where $a(.)$ is the nonlinear mapping function produced by the MLP, while $\boldsymbol{w}$ and $\boldsymbol{x}$ are the parameters (i.e. weights) and state (i.e. input) vector respectively. As in the linear-in-parameter technique, the nonlinear

estimation requires the gradient vector $\nabla\psi[\boldsymbol{w}(k)]$ to be derived, that is

$$\nabla\psi[\boldsymbol{w}(k)] = \frac{\partial}{\partial \boldsymbol{w}}\boldsymbol{a}[\boldsymbol{w}(k), \boldsymbol{x}(k)] \tag{2.38}$$

where

$$\begin{aligned}
\nabla\psi[w_{cj}^{L}(k)] &= \frac{\partial E_c(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial w_{cj}^{L}(k)} \\
\nabla\psi[w_{cij}^{NL}(k)] &= \frac{\partial E_c(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial y_j(k)} \cdot \frac{\partial y_j(k)}{\partial g_j(k)} \cdot \frac{\partial g_j(k)}{\partial w_{cij}^{NL}(k)} \\
\nabla\psi[w_{aj}^{L}(k)] &= \frac{\partial E_a(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial u(k)} \cdot \frac{\partial u(k)}{\partial w_{aj}^{L}(k)} \\
\nabla\psi[w_{aij}^{NL}(k)] &= \frac{\partial E_c(k)}{\partial Q(k)} \cdot \frac{\partial Q(k)}{\partial u(k)} \cdot \frac{\partial u(k)}{\partial z_j(k)} \cdot \frac{\partial z_j(k)}{\partial f_j(k)} \cdot \frac{\partial f_j(k)}{\partial w_{aij}^{NL}(k)}
\end{aligned} \tag{2.39}$$

The Gauss-Newton Hessian, $R(k)$, matrix approximation can be estimated recursively at each iteration by

$$R(k) = \alpha(k)R(k-1) + [1-\alpha(k)](\nabla\psi[\boldsymbol{w}(k)]\nabla\psi^{T}[\boldsymbol{w}(k)]) \tag{2.40}$$

where α is called the forgetting factor and controls the rate at which $R(k)$ adapts to new inputs. It is usually set to $0.975 < \alpha < 1.0$. The weight update is then given by

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + R(k)^{-1}(\nabla\psi[\boldsymbol{w}(k)])e(k) \tag{2.41}$$

where $e(k)$ is the temporal difference error, either $e_c(k)$ for the Critic Network or $e_a(k)$ for the Action Network (Eq. 2.36). This recursive formulation is rarely used directly due to the $O(N_w^3)$ computational complexity of the inverse calculation which has to be performed at each iteration [34]. Instead the matrix inversion lemma,

$$(A+BC)^{-1} = A^{-1} - A^{-1}B(1+CA^{-1}B)^{-1}CA^{-1}, \tag{2.42}$$

is used to compute the inverse of $R(k)$. This leads to the following weight update procedure, referred to as the recursive prediction error (RPE) algorithm ([9],[10]):

$$P(k) = \frac{1}{\alpha}[P(k-1) - P(k-1)(\nabla\psi[\boldsymbol{w}(k)])S^{-1}(k)(\nabla\psi^{T}[\boldsymbol{w}(k)])P(k-1)] \tag{2.43}$$

$$S(k) = \alpha(k) + (\nabla\psi^{T}[\boldsymbol{w}(k)])P(k-1)(\nabla\psi[\boldsymbol{w}(k)]) \tag{2.44}$$

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + P(k)(\nabla\psi[\boldsymbol{w}(k)])e(k) \tag{2.45}$$

The matrix $P(k)$ ($= R(k)^{-1}$) here can be interpreted as the covariance matrix of the weight estimate $\boldsymbol{w}(k)$. If the input signals to the plant are not well excited, the covariance matrix $P(k)$ tends to become large, a phenomenon known as *covariance wind-up* ([9],[10]). A constant trace adjustment is usually applied to the covariance matrix, as suggested by Salgado *et. al* [44], to overcome this problem. Thus

$$P(k) = \frac{\tau}{trace[P(k)]}P(k) \qquad \tau > 0 \tag{2.46}$$

and τ is a positive scalar normally set to value of one [1].

Ljung and Söderström [31], suggested that by using a time varying forgetting factor, $\alpha(k)$, rather than a fixed value with $\alpha(k) < 1$ at the beginning, $P(k)$ convergences faster from its initial value and $\alpha(k) \to 1$ as $k \to \infty$ will provide stability. This is achieved by using the following update rule for $\alpha(t)$:

$$\alpha(k) = \bar{\alpha}\alpha(k-1) + (1-\bar{\alpha}) \tag{2.47}$$

where $\bar{\alpha}$ is a scalar (<1) which determines the rate of convergence of $\alpha(k)$ to 1. According to Gunnarson ([17],[18]), the recursive Levenberg-Marquardt (RLM) algorithm is simply the regularised form of the recursive prediction error (RPE) method. Thus, RLM is obtained by incorporating a regularisation term in the Hessian update expressed in equation 2.40, giving

$$R(k) = \alpha(k)R(k-1) + [1-\alpha(k)]\left\{(\nabla\psi[\boldsymbol{w}(k)]\nabla\psi^{T}[\boldsymbol{w}(k)]) + \rho I_{N_w}\right\} \tag{2.48}$$

Unfortunately, its now impractical to directly apply the matrix inversion lemma. Ngia *et al.* [34,35] proposed adding the regularisation term ρ to one diagonal element of $R(k)$ at a time as a solution to this problem. This corresponds to rewriting Equation 2.48 as

$$R(k) = \alpha(k)R(k-1) + [1-\alpha(k)]\left\{(\nabla\psi[\boldsymbol{w}(k)]\nabla\psi^{T}[\boldsymbol{w}(k)]) + \rho Z_{N_w}\right\} \tag{2.49}$$

where Z_{N_w} is an $N_w \times N_w$ zeroes matrix, except with one of its diagonal elements set to one. The diagonal element z_{ii} set equal to 1 changes from iteration to iteration as determined by the following expressions,

$$z_{ii} = 1 \text{ when } i = k \bmod(N_w) + 1 \text{ and } k > N_w \tag{2.50}$$

$$z_{ii} = 0 \text{ otherwise} \tag{2.51}$$

With this modification, equation 2.49 can be rewritten as

$$R(k) = \alpha(k)R(k-1) + [1-\alpha(k)][\Omega(k)\Lambda(k)^{-1}\Omega^T(k)] \tag{2.52}$$

where $\Omega(k)$ is a N_w x 2 matrix with the first column containing $\nabla\psi[\boldsymbol{w}(k)]$ and the second column consisting of a N_w x 1 zero vector with one element set to 1 in accordance with 2.50 and 2.51 above, that is.

$$\Omega^T(k) = \begin{bmatrix} & \nabla\psi[\boldsymbol{w}(k)] & \\ 0 \ \ldots & 1 & \ldots \ 0 \end{bmatrix} \quad \text{and } \Lambda(k)^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{2.53}$$

$$position = k \bmod(N_w) + 1$$

The matrix inversion lemma can now be applied to equation 2.52 which leads to the recursive Levenberg-Marquardt (RLM) formulation [34].

$$S(k) = \alpha(k)\Lambda(k) + \Omega^T(k)P(k-1)\Omega(k) \tag{2.54}$$

$$P(k) = \frac{1}{\alpha(k)}[P(k-1) - P(k-1)\Omega(k)S^{-1}(k)\Omega^T(k)P(k-1)] \text{ with } P(k) = \frac{1}{trace[P(k)]}P(k) \tag{2.55}$$

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + P(k)(\nabla\psi[\boldsymbol{w}(k)])e(k) \tag{2.56}$$

$S(k)$ is now a 2×2 matrix, which is much more cost effective to invert than the $N_w \times N_w$ matrix that arises when the matrix inversion lemma is applied to equation 2.48.

The overall training procedure for the ADAC is summarised in Figure 2-9. Training begins with the weights being initialised randomly before each run. At time *k*, the Action Network and Critic Network both receive the input state vector, $\boldsymbol{x}(k)$. The Action Network outputs a control action, $u(k)$, to the plant. At the same time the Critic Network outputs $Q(k)$ and it is stored. At time k+1, both the Action and Critic network receives the next state vector, $\boldsymbol{x}(k+1)$ and produces, their corresponding outputs - $u(k+1)$ and $Q(k+1)$ respectively. The reward $r(k+1)$ is obtained based on the outcome of the control action, i.e. $r = 0$ when it is a success or

$r = -1$ when it is a failure. The Critic Network weights are updated once this reward value is obtained. Then the Action Network weights are updated. Once the updates are done, the cycle is repeated until the stopping criteria has been met.

Action Dependent Adaptive Critic Training Summary

1. At the start of every run initialise all network weights to random values in the range [-1.0,1.0].
2. At time *t*, retrieve the state vector, ***x***(*k*), from the process or plant being controlled.
3. Generate a control action, *u*(*k*) and apply it to the plant.
4. Generate and store the Critic Network output $Q(k)$ (to be used as "$Q(k-1)$" to calculate the cost function).
5. Retrieve the new state vector, ***x***(*k*+1), from plant and generate the next control action, *u*(*k*+1).
6. Generate the next Critic Network's output $Q(k+1)$.
7. Calculate the external reinforcement, *r*(*k*+1) resulting from the control action.
8. Adjust the learning rates, β_c and β_a.
9. Adjust the Critic Network and the Action Network weights using Eq. 2.22 - 2.34 for TD-SBP, or using Eq. 2.54 - 2.56 for TD-RLM
10. Repeat from Step 3 until the stopping criteria has been met.

Figure 2-9 Summary of the Action Dependent Adaptive Critic Algorithm

In the next section the proposed TD-RLM algorithm will be evaluated in simulation on the well known inverted pendulum case study and its performance compared to the TD-SBP algorithm used by Si and Wang [47].

3.0 Inverted Pendulum

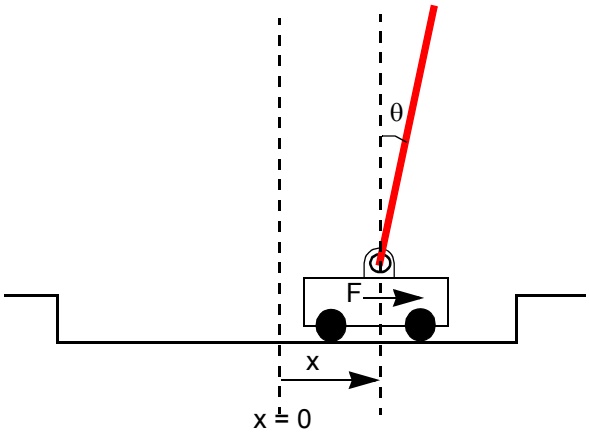

Figure 3-1 *The inverted pendulum system*

The inverted pendulum or pole-balancing task is representative of the inherently unstable, multiple-output, dynamic systems present in many balancing situations, like two-legged walking robots and the aiming of a rocket thruster and is frequently used as a benchmark for modern and classical control-engineering techniques. The inverted pendulum problem, depicted in Figure 3-1, has historical importance for reinforcement learning as it was one of the first successful applications of an algorithm based on model-free action policy estimation, as described in 1983 by Barto *et al*. in their pioneering paper [6]. Systems that learn to control inverted pendulums were first developed over thirty years ago by Michie and Chambers [33] and have been the subject of much research since then. See for example [1], [2], [3],[11], [15], [16],[20], [22], [45], [47],[58] and [59].

Control-engineering techniques involve detailed analyses of the system to be controlled. When the lack of knowledge about a task precludes such analyses, a control system must adapt as information is gained through experience with the task. To investigate the use of learning methods for such cases, it is assumed that very little is known about the inverted pendulum system, including its dynamics. The system is viewed as a black box generating as output the system's state and accepts as input a control action.

The inverted pendulum task considered here involves balancing a pole hinged to the top of a wheeled cart that travels along a track, as shown in Figure 3-1. The control objective is to apply a sequence of right and left forces of fixed magnitude to the cart so that the pole balances and the cart does not hit the end of the

track (for this simulation a -10N and 10N correspond to the left and right forces respectively). A zero magnitude force is not permitted. The state of the cart-pole system has to be remain outside certain regions of the state space to be considered successfully controlled. There is no unique solution and any state space trajectory that does not pass through these regions is considered acceptable. The only information provided regarding the control goal is the reinforcement signal, $r(k)$, which signals a failure when either the pole falls outside $\pm 12^{o}$ or the cart hits the bounds of the track at $\pm$ 2.4 m.

The cart-pole simulation model used in this study is defined by the equations

$$\ddot{\theta}(t) = \frac{g\sin\theta(t) - \cos\theta(t)\left[\frac{-F(t) + ml\dot{\theta}^2(t)\sin\theta(t) + \mu_c sgn[\dot{x}(t)]}{m_c + m}\right] - \frac{\mu_p\dot{\theta}(t)}{ml}}{l\left[\frac{4}{3} - \frac{m\cos^2\theta(t)}{m_c + m}\right]} \tag{3.1}$$

$$\ddot{x}(t) = \frac{F(t) + ml[\dot{\theta}^2(t)\sin\theta(t) - \ddot{\theta}(t)\cos\theta(t)] - \mu_c sgn[\dot{x}(t)]}{m_c + m} \tag{3.2}$$

$$sgn(x) = \begin{cases} 1, \text{ if } x > 0 \\ 0, \text{ if } x = 0 \\ -1, \text{ if } x > 0 \end{cases} \tag{3.3}$$

This includes all the nonlinearities and reactive forces of the physical system such as friction. The system is constrained to move within the vertical plane. Here $x(t)$ and $\theta(t)$ are the horizontal position of the cart and the angle of the pole, respectively and

l, the length of the pole	= 0.5m
m, the mass of the pole	= 0.1kg
m_c, the mass of the pole and cart	= 1.1kg
F, the magnitude of the force	= $\pm$10N
g, the acceleration due to gravity	= 9.8 ms^{-2}
μ_c, friction of cart on track coefficient	= 5 x 10^{-4}
μ_p, friction of pole on cart coefficient	= 2 x 10^{-6}

This set of nonlinear differential equations, was solved numerically using fourth-order Runge-Kutta in the simulation and sampled every 0.02 seconds to produce four discrete state variables defined as follows:

$x(k)$ = the horizontal position of the cart, relative to the track, in metres,
$\dot{x}(k)$ = the horizontal velocity of the cart, in metres/second,
$\theta(k)$ = the angle between the pole and vertical, in degrees, clockwise being positive,
$\dot{\theta}(k)$ = the angular velocity of the pole, in degrees/second.

3.1 Inverted Pendulum Control Results

The ADAC used to implement the inverted pendulum controller is illustrated in Figure 3-2. Both the Action and Critic networks were implemented as single hidden layer MLP neural networks. Each network had 6 neurons in their hidden layer. Thus the Action network was a 4-6-1 MLP architecture and the Critic Network was a 5-6-1 architecture.

The simulation studies were based on 100 runs, each consisting of 10 trials, during which the Action Network had to control the inverted pendulum within set boundaries. Here the external reinforcement signal was defined as

$$r(k) = \begin{cases} -1, \text{ If } |\theta| > 12° \text{or } |x| > 2.4m \\ 0, \text{ otherwise} \end{cases} \tag{3.4}$$

The controller was considered to be successful if it managed to balance the inverted pendulum for 6 x 10^5 time steps of 0.02s each (i.e. for 3 hours 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 network weights initialised randomly. Figure 3-3 depicts the result of a typical successful run of the ADAC controller.

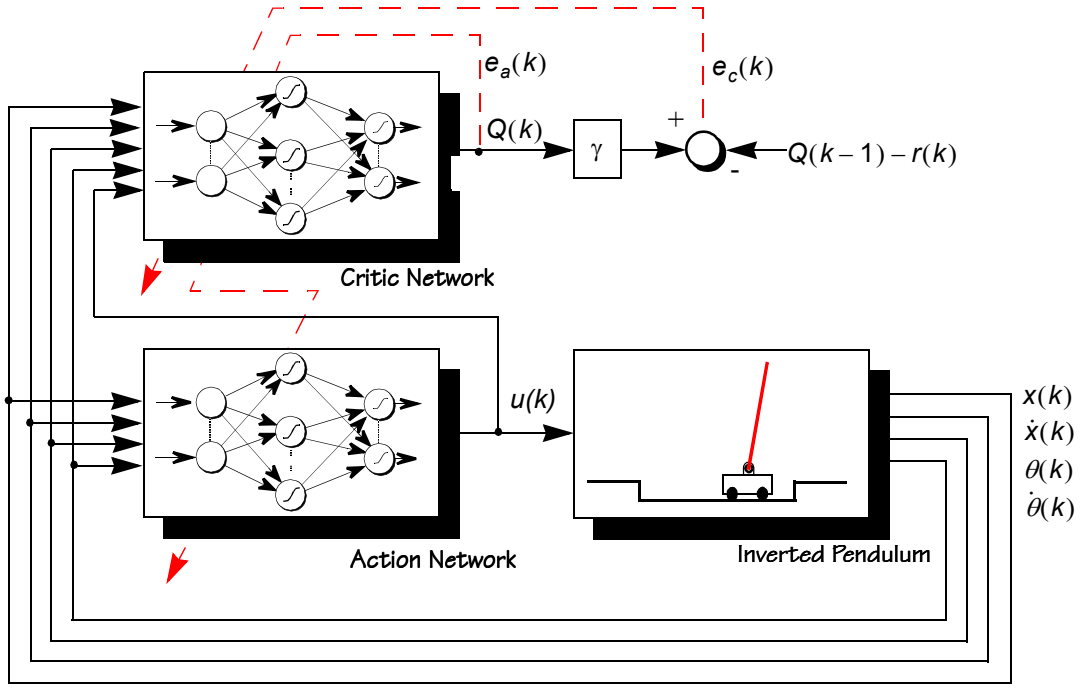

Figure 3-2 *The ADAC control strategy for the inverted pendulum system*

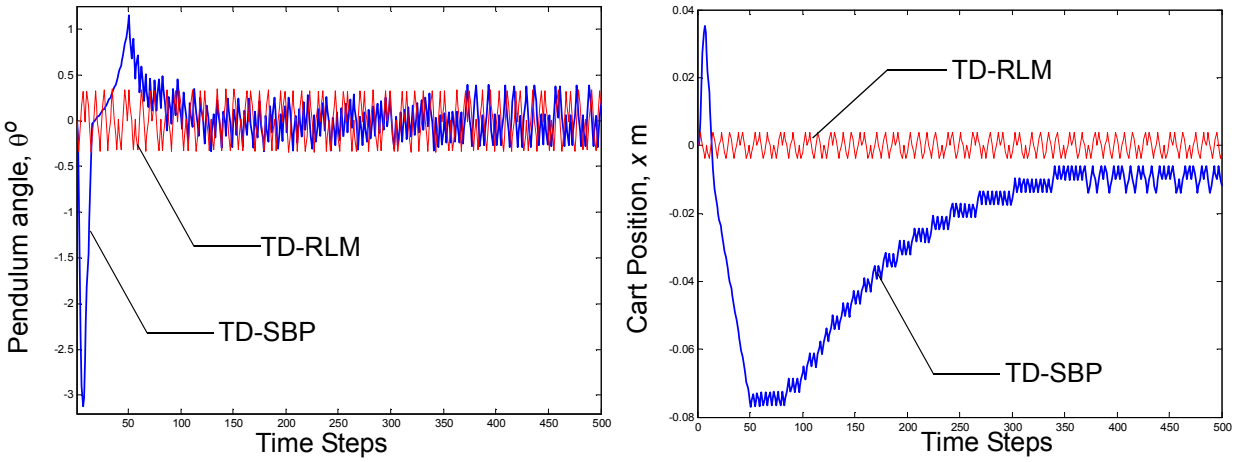

Figure 3-3 *An example of a typical pendulum angle and cart position trajectory for a successful run using TD-SPB and TD-RLM*

Figure 3-3 clearly shows that the TD-RLM algorithm convergences almost immediately compared to the ADAC which was trained using TD-SBP. Also note that the ADAC controller balances the pendulum, using a force, of a fixed magnitude and alternating sign applied to the cart (see Figure 3-4). This bang-bang control leads to the zig-zag oscillation observed in the graphs.

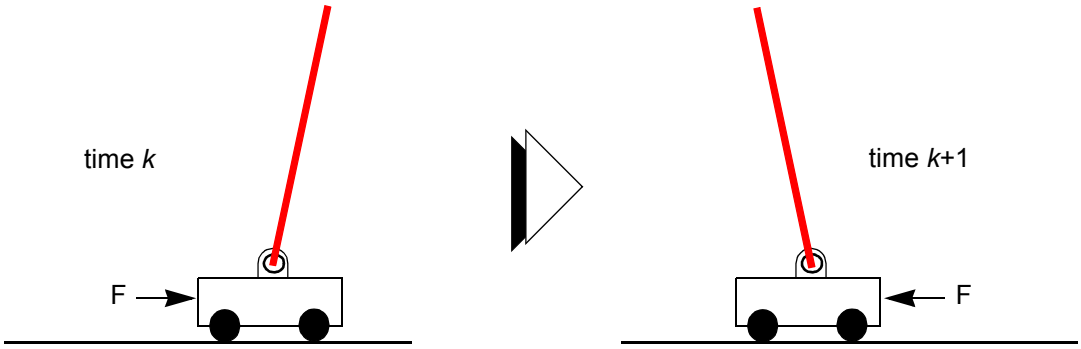

Figure 3-4 *Forces applied to the cart to balance the inverted pendulum between a sample interval*

The performance of both training algorithms was measured in terms of the frequency of successful runs and the average computational time for a successful run. The actual computational cost was also considered as a function of Action and Critic network sizes, to get a better measure of overall performance, in terms of the training efficiency.

Figure 3-5 shows that the TD-RLM training algorithm gave the best overall performance in terms of the frequency of successful runs. It can also be seen, that the original TD-SBP algorithm by Si and Wang [47] was quite poor in terms of the number of successful runs, being at times 50% lower than the alternative second-order method.

Furthermore, as shown in Figure 3-6, the TD-SBP implementation also proved less efficient compared to TD-RLM, in terms of average computation time per successful run, with the second order method being the most consistent for different network sizes. The overall average computation time per successful run as the ADAC network sizes varied from 2 to 10 hidden neurons was 459.75 secs for TD-SBP and 350.41 secs for TD-RLM.

Figure 3-7 plots the time variations in the squared training error for the Critic Network obtained with each algorithm. These illustrate the speed of convergence of the neurocontroller weights. Again, the zig-zag oscillation is due to the nature of the bang-bang control strategy employed.

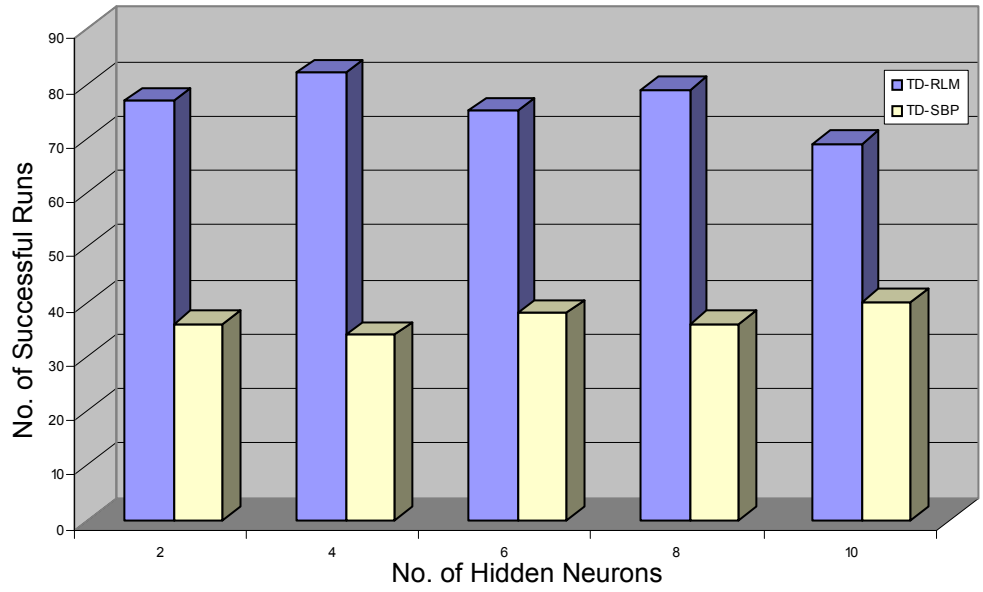

Figure 3-5 *Variation in number of successful runs with the size of the ADAC Networks*

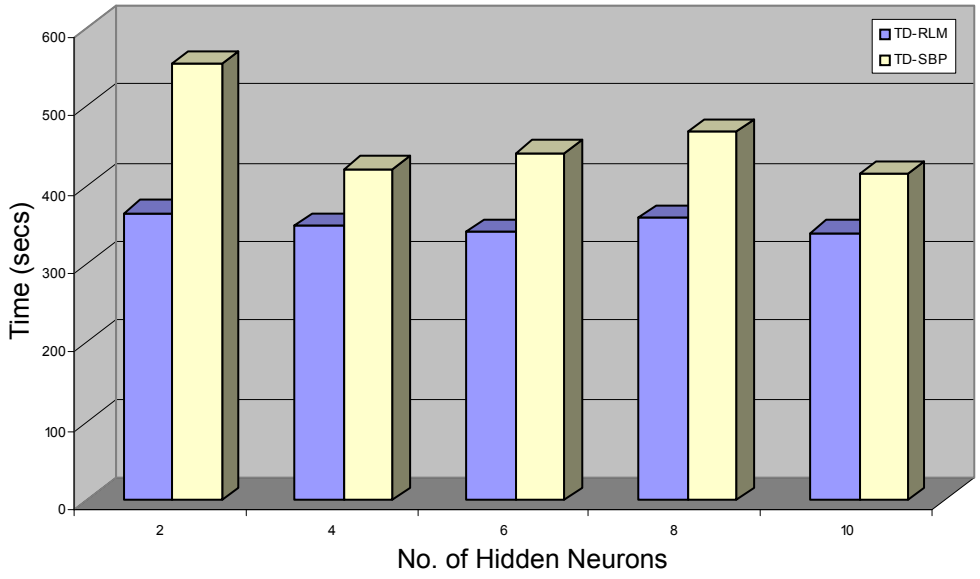

Figure 3-6 *Variation in averaged computation time per successful run with the size of the ADAC Networks*

Further evidence of the superior convergence speed produced by second-order training is provided in Figure 3-8. This shows the variation in the first component of the weight vector for the Action Network and the Critic Network, $w_{a1,1}^{NL}$ and $w_{c1,1}^{NL}$ respectively, with each algorithm.

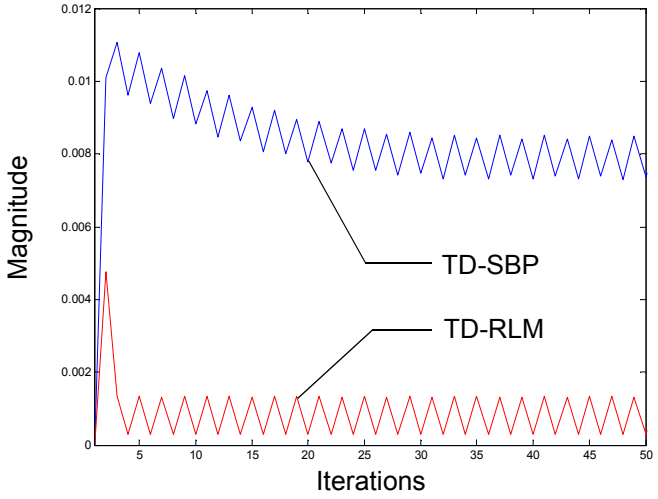

Figure 3-7 *Comparison of the training cost E_c of the Critic Network obtained with TD-RLM and TD-SBP.*

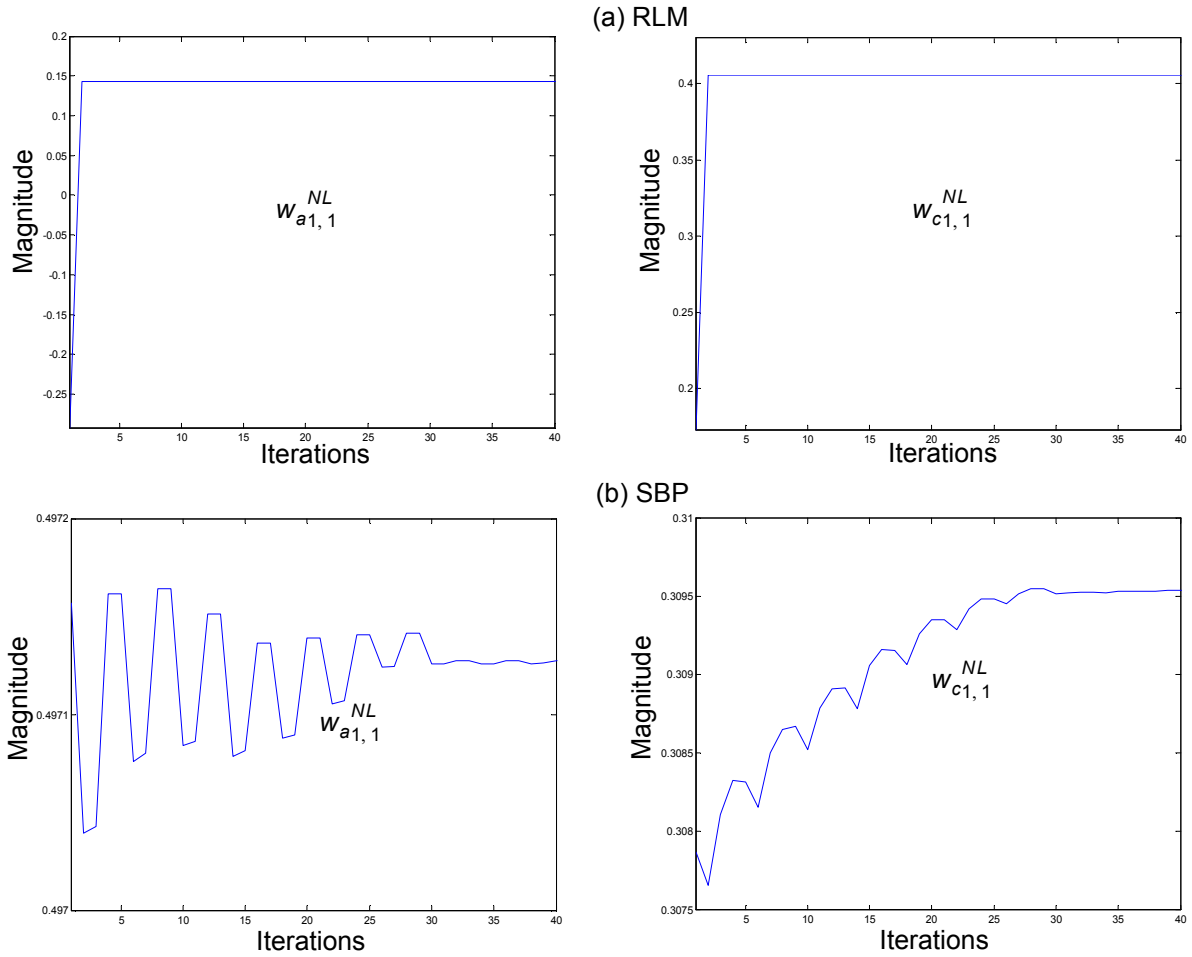

Figure 3-8 *The trajectory of the first components of the weight vector of the Action Network, $w_{a1,1}^{NL}$ (left) and of the Critic Network, $w_{c1,1}^{NL}$ (right) for (a) TD-RLM and (b) TD-SBP*

4.0 Ring Grinding Process

The second application is the ADAC modelling and control of an industrial ring grinding process used in the manufacture of disk drive platters. Here, aluminium subtrate disks are ground in batches of twelve between two grindstones, as shown in Figure 4-1 and 4-2. The stones can be moved apart to allow loading and unloading of the disks using a pick-and-place unit. During operation the grindstones are rotated in opposite directions with pressure applied to the upper one. This causes the subtrate disks between them to rotate, thereby ensuring uniform grinding of their surfaces. The rate at which the disks are ground, called the removal rate, is the critical variable. It varies depending on a number of parameters including stone wear, exerted pressure, lubricant viscosity and coolant flow rate. The initial thickness of the disks also varies, although the disks in any one batch are sorted to be approximately the same thickness. The thickness of one disk from each batch is measured before the batch is ground. The system controller determines the actual removal rate from the previous batch and estimates the current value of removal rate using a proprietary control law. It predicts how much material has to be removed by subtracting the target thickness from the input thickness and then calculates the necessary grinding duration for the current batch.

When the grinding is completed, the selected disk is measured again. If it is within specification, then the whole batch is passed. If the disk is too thick (above the upper specification limit), the disks are ground again (i.e. reworked) but if the disk is too thin (below the lower specification limit), the batch is rejected. When a grindstone is newly installed (i.e. replaced due to wear), the pressure is initially set to a low value and then gradually increased to an upper limit to counteract the stone deterioration, which in turn increases the removal rate. The removal rate subsequently decreases until a stage is reached where it is so low that the grindstone has to be resurfaced which is done by slicing off the worn part of the grindstone. Once re-installed the whole process is repeated.

Various process variables are logged for each grind cycle as part of the company's own process performance monitoring procedure. These include the current removal rate, the pressure between the grindstones, and the cumulative cycle time. Cumulative cycle time is logged as it is an indication of wear

and aging of the grindstones, which in turn impacts on the removal rate. A summary of these variables and the identifiers used for them in this chapter is provided in Table 4.1.

The grindstone data used in this investigation was detrended and normalised to lie within the interval [-1,1]. A sample of the data showing the typical variations found in all the variables is plotted in Figure 4-3.

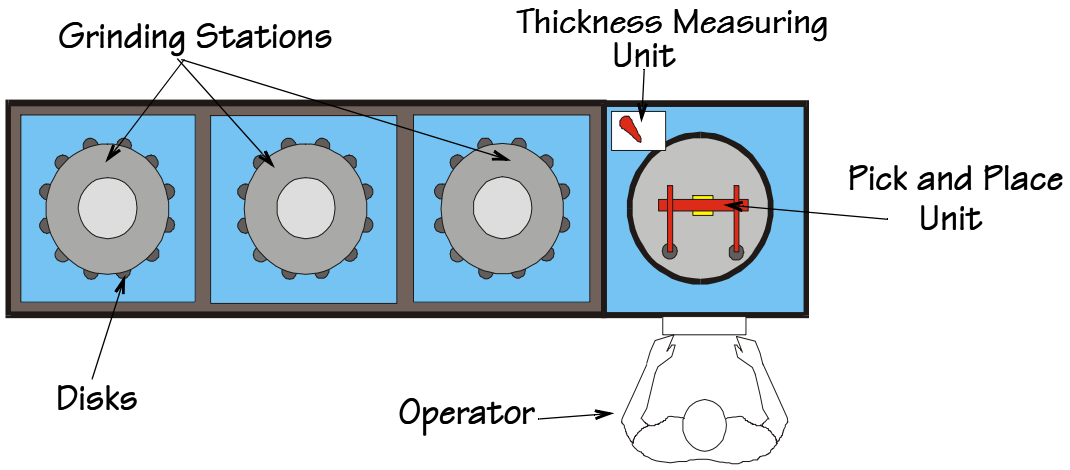

Figure 4-1 *Layout of the ring grinding process*

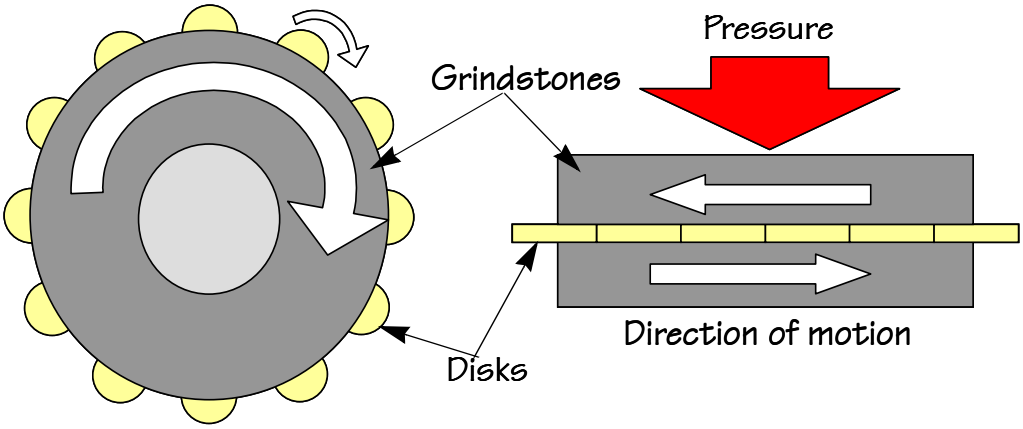

Figure 4-2 *The ring grinding process*

Variables	Definition
Removal Rate, $rr(k)$	Rate of material removal from a disk during the grind cycle. Units are in microinch per min.
Previous Removal Rate, $rr(k\text{-}1)$	Removal rate from the previous grind cycle.
Cycle Time, $c(k)$	Grind cycle time. Units are in seconds.
Cumulative Cycle Time, $cct(k)$	Sum of all previous cycle times since the grindstone was last resurfaced.
Pressure, $p(k)$	Pressure between the grindstones. Units are in p.s.i.
Loading Thickness, $T_L(k)$	Thickness of the disk before the grinding process begins. Units are in mil(s).
Unloading Thickness, $T(k)$	Thickness of the disk after the completion of the grinding process. Units are in mil(s).
Target Thickness, $T_{SP}(k)$	Desired thickness required for the each grind cycle. Units are in mil(s).
Upper Control Limit, $UCL(k)$	Upper control thickness limit specification. Units are in mil(s).
Lower Control Limit, $LCL(k)$	Lower control thickness limit specification. Units are in mil(s).

Table 4.1: *Grinding Process Variables*

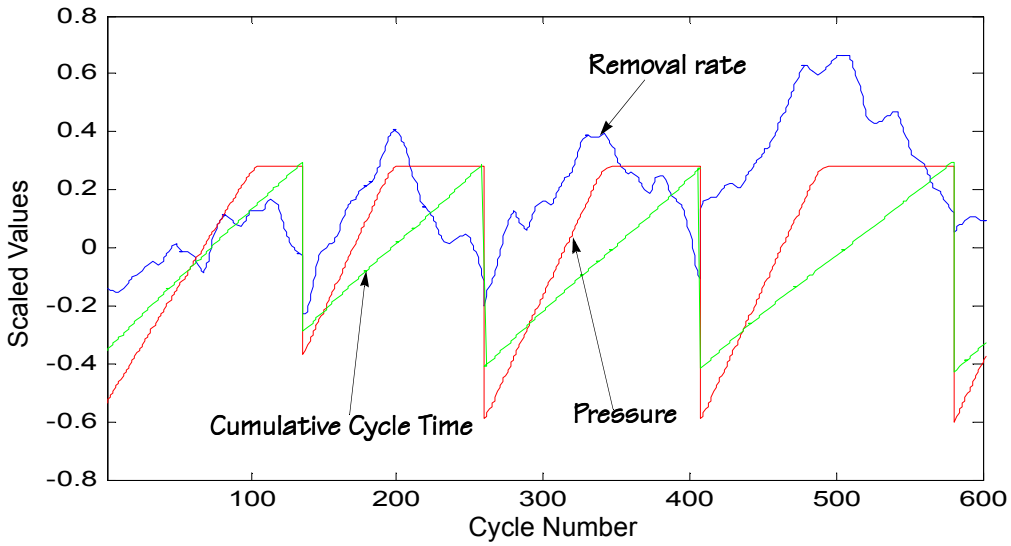

Figure 4-3 *Variables used in modelling the ring grinding process*

The main aim here is to achieve accurate thickness control in order to minimise the number of out-of-specification disks produced by the grinding process. This process optimisation can be achieved through manipulation of the grind cycle time as illustrated in Figure 4-4. Neural network based direct inverse control

has been shown to provide an effective solution to this problem [14] and was therefore chosen as the basis for the ADAC investigation. The ADAC framework was consider for two elements of the controller design, namely developing a process model and fine-tuning the final controller. A process model is needed as this forms the basis for the direct inverse controller implementation. The recommended model is one which predicts the removal rate, for each grind cycle, on the basis of the current state of the process [14].

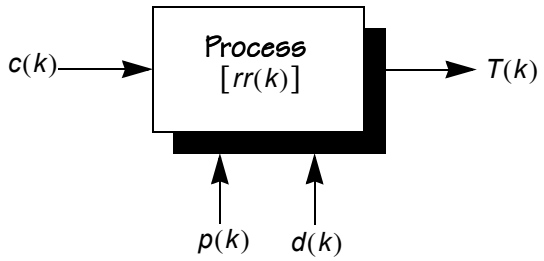

Figure 4-4 The ring grinding process block diagram

The existing proprietary controller was used as a reference for evaluating the performance of the model and subsequent controller design. Since the proprietary controller was a model free implementation, it did not generate an explicit removal rate prediction, $\hat{rr}(k)$. Rather, this was inferred from the generated cycle time, *c*(*k*), as

$$\hat{rr}(k) = \frac{T_L(k) - T_{SP}(k)}{c(k)} \tag{4.1}$$

where $T_L(k)$ is the loading thickness and $T_{SP}(k)$ is the setpoint or target thickness. Note that the actual removal rate, *rr*(*k*), was obtained by replacing $T_{SP}(k)$ by the measured unloading thickness, $T(k)$, in Eq. 4.1, that is

$$rr(k) = \frac{T_L(k) - T(k)}{c(k)} \tag{4.2}$$

Figures 4-5 compares the predicted removal rate, $\hat{rr}(k)$, with the actual removal rate, *rr*(*k*), over the life of a typical grindstone. The accuracy of the prediction is measured in terms of the percentage normalised mean prediction error (MPE), defined as

$$MPE = \frac{1}{n}\sum_{j=1}^{n} \frac{\left|rr(k) - \hat{rr}(k)\right|}{\sigma(k)} \times 100\% \tag{4.3}$$

where $\sigma(k)$ is the standard deviation of $rr(k)$. In this case the MPE for the grindstone was 6.8%.

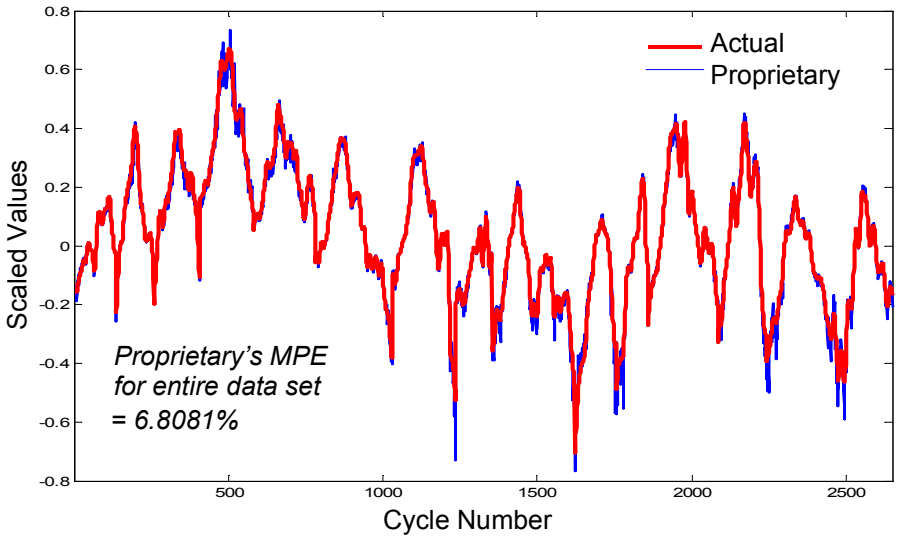

Figure 4-5 *The removal rate prediction from the proprietary scheme*

4.1 Model Development

The process model to be developed here predicts the grindstone removal rate and this is used to calculate the cycle time for the grinding machine. Accurate prediction of removal rate will therefore lead to an improved cycle time estimate for the grind process. In the ADAC framework, the Action Network is trained to form the process model.

Based on previous experience [14], it was decided to incorporate error feedback to compensate for the low-frequency offsets in the Action network prediction, in order to further enhance the accuracy of the process model. This "predict-correct" technique uses past plant outputs and the corresponding model predictions to generate a correction to the current estimate $\hat{rr}^*(k)$ and successful applications have been reported in Rovlak and Corlis [43], Willis et al. [60], Lightbody [28] and Irwin et al. [24]**.** The predict-correct scheme is implemented as follows

$$\hat{rr}^*(k) = \hat{rr}(k) + \frac{1}{N}\sum_{j=1}^{N}[rr(k-j) - \hat{rr}(k-j)] \qquad (4.4)$$

A first order, predict-correct term was incorporated into the Action network predictor, as shown in Figure 4-

6. The complete ADAC identification strategy is then as shown in Figure 4-7.

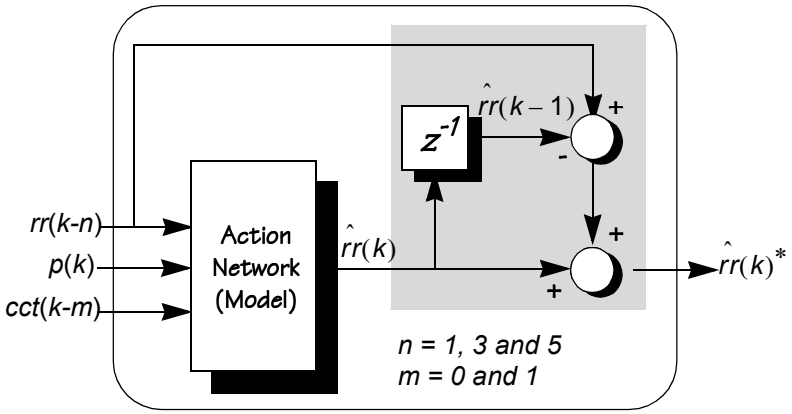

Figure 4-6 *The augmented Action network prediction model for the grinding process (Model + PC)*

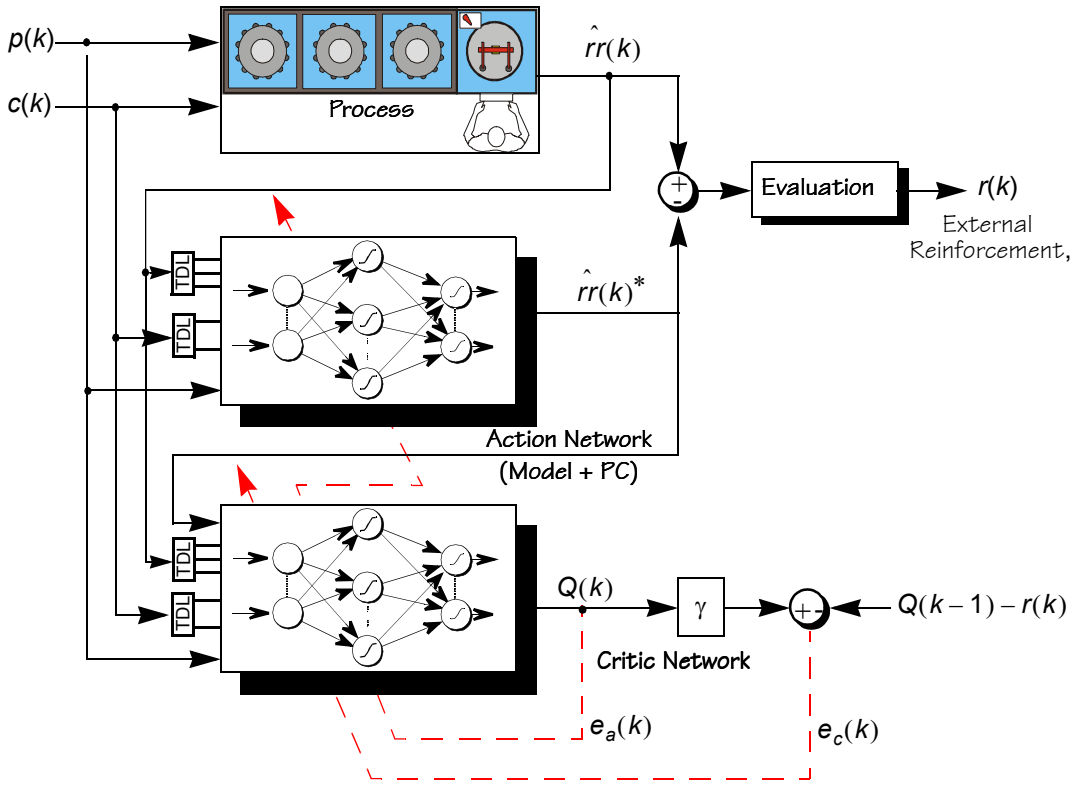

Figure 4-7 *Schematic of modelling strategy using the ADAC*

For this study a nonlinear ARX modelling strategy was employed where the removal rate was estimated as a function of previous removal rates, *rr*(*k*-1), *rr*(*k*-3) and *rr*(*k*-5), current pressure, *p*(*k*), and the current and past cumulative cycle times, *cct*(*k*) and *cct*(*k*-1). Thus, the Action Network was trained to learn the unknown mapping

$$\hat{rr}(k) = f[rr(k-1),\ rr(k-3),\ rr(k-5),\ p(k),\ cct(k),\ cct(k-1)] \tag{4.5}$$

The goal was to minimise the absolute tracking error between the desired removal rate, $rr(k)$, and the predicted, $rr^*(k)$. The external reinforcement signal, r(k), for the ADAC model was chosen as

$$r(k) = \begin{cases} -1, & |\text{Prediction Error}| \geq 5 \times 10^{-4} \\ 0, & otherwise \end{cases} \tag{4.6}$$

After experimenting with different architectures, the ADAC networks that were found to produce the minimum MPE error used 5 hidden neurons (i.e a 6-5-1 MLP for the Action network and a 7-5-1 MLP for the Critic network). Figure 4-8 compares the ADAC model obtained for the grindstone with the corresponding proprietary model and clearly shows the benefit of nonlinear modelling using the ADAC framework as the MPE has been reduced by 42% compared to the proprietary model.

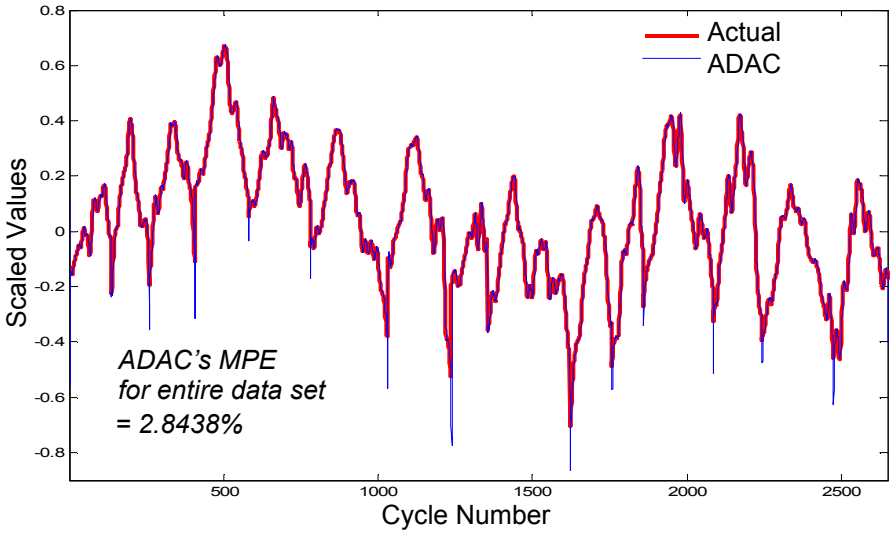

Figure 4-8 *The removal rate prediction from the ADAC scheme*

4.2 Disk Thickness Control Results

The model identified previously was used to provide an accurate estimate of $rr(k)$ at each iteration to produce the open-loop thickness controller depicted in Figure 4-9. In fact this is a direct inverse control implementation, which can be seen as follows. First, note that the removal rate model can also be used to generate a $c(k)$-to-$T(k)$ forward process model, as shown in Figure 4-10. Lee and Shin [27] pointed out

that, this particular formulation allows the inverse plant model to be obtained without having to invert a ANN model, as is usually the case with a neural-control scheme [23]. Thus, Figure 4-9 represents an exact inverse of the forward process model and therefore a direct inverse controller. The final complete direct inverse model control scheme is shown in Figure 4-11.

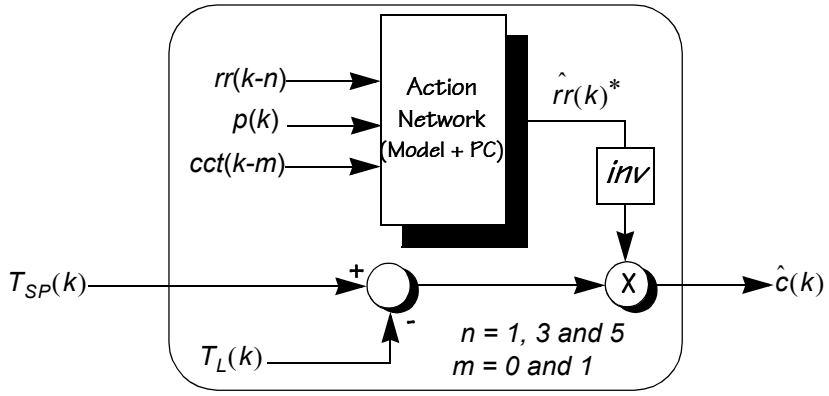

Figure 4-9 *Open-loop control using the Action Network removal rate predictor*

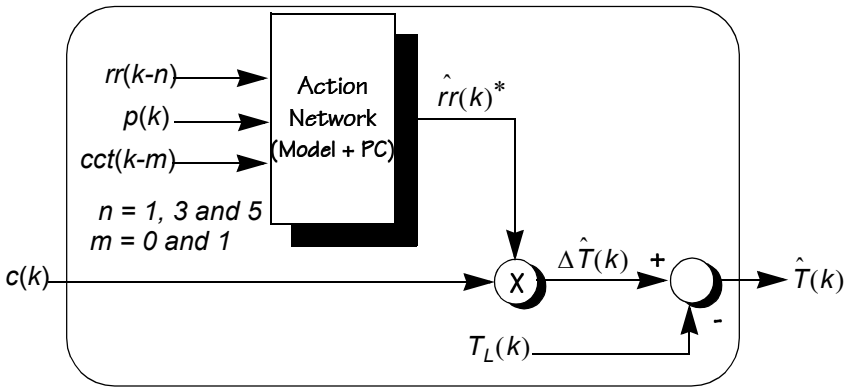

Figure 4-10 *Forward process ADAC model*

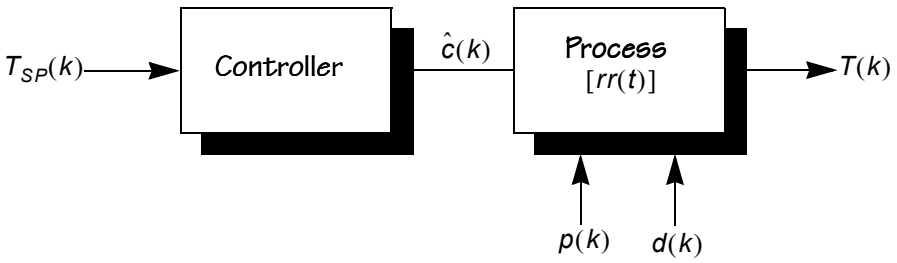

Figure 4-11 *Open-loop inverse model control using the ADAC modelling method of the grinding process*

The ADAC-model direct inverse controller can be applied to the process as a fixed parameter controller. Alternatively it can be fine tuned on-line within an ADAC control framework, with the reinforcement signal now defined in terms of control specifications, namely the upper and lower control limits (UCL and LCL) for the unloading thicknesses of the disks. This gives

$$r(k) = \begin{cases} -1, \text{ UCL} < \text{ULT} < \text{LCL} \\ 0, otherwise \end{cases} \tag{4.7}$$

Figure 4-12 shows the unloading thickness prediction obtained with the resulting ADAC control strategy while Table 4.2 provides a comparison with the fixed parameter ADAC model based controller and the proprietary controller.

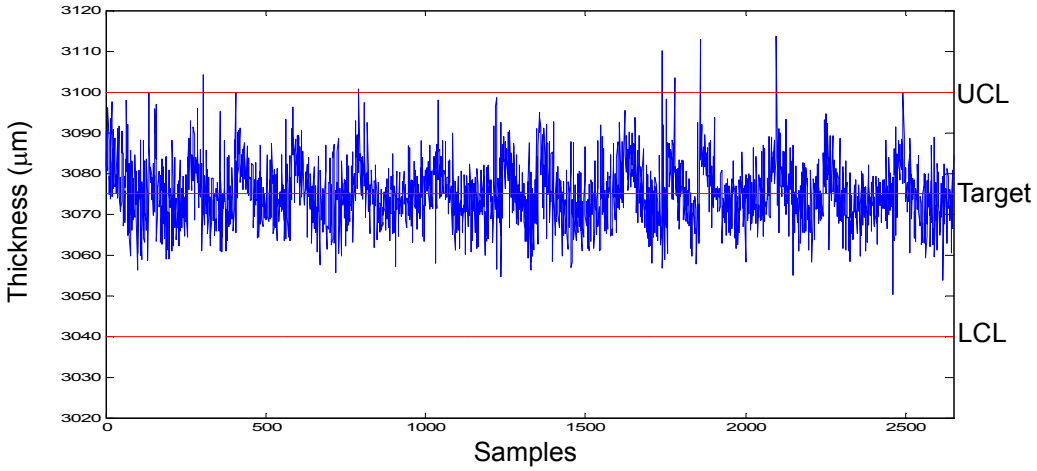

Figure 4-12 *Unloading thickness prediction of the ADAC controller*

Model	Target Thickness	Unloading Thickness Mean	Unloading Thickness Variance	Number of Rejects
Proprietary		3075.85	7.095	12
ADAC-mod.	3075	3076.12	6.71	10
ADAC-cont.		3076.28	6.407	8

Table 4.2: *Performance comparison between the actual and the ADAC schemes*

It can be seen that the ADAC model direct inverse controller (ADAC-mod) and the online tuned ADAC controller (ADAC-cont.) both outperform the proprietary scheme with the tuned ADAC controller yielding a 33.33% reduction in the number of rejects. Figures 4-13 compares the unloading thickness distribution of

the proprietary control scheme with that obtained from the ADAC controller and clearly show than tighter thickness control was achieved with the ADAC control scheme.

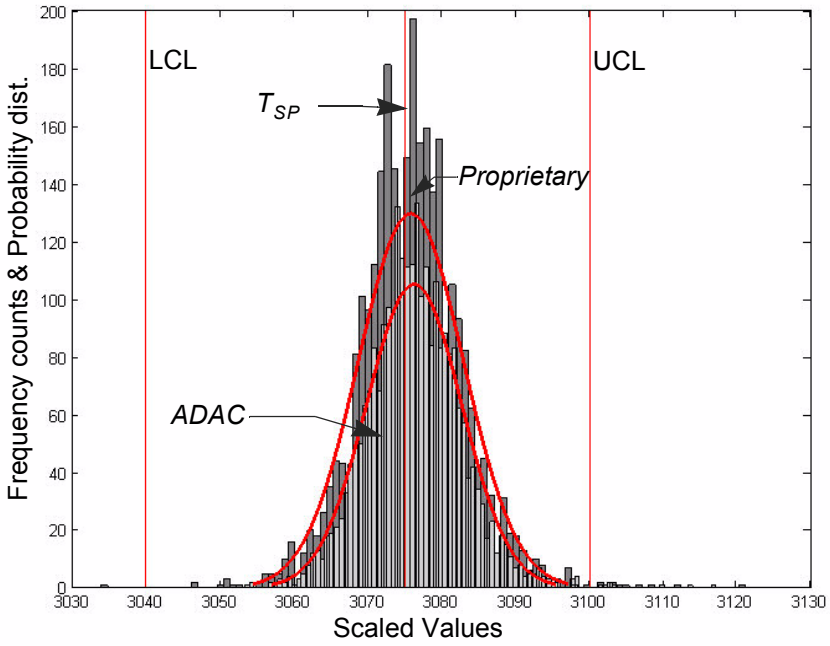

Figure 4-13 *Unloading thickness distribution plot of the ADAC method compared to the proprietary scheme*

5.0 Conclusions

This chapter extends the model-free action dependent adaptive critic (ADAC) of Si and Wang by presenting a fully online, intelligent neurocontroller which avoids the necessity to store plant data during a successful run. In particular, the potential limitations of their stochastic backpropagation training, in terms of poor convergence and parameter shadowing, are avoided by introducing a modified version of recursive Levenberg-Marquardt (RLM) called the temporal difference RLM (TD-RLM) algorithm. This was demonstrated in a initial simulation study on inverted pendulum control.

The performance of the new ADAC scheme, for both identification and control, has been validated using data from an actual industrial grinding process used in the manufacture of aluminium substrates for disk drives. The results suggest that the ADAC can achieve a 33% reduction in rejects compared to a proprietary controller. The authors believe that this is one of the first reported industrial applications of this emerging technology.

For identification the ADAC still requires the incorporation of the predict correct strategy to achieve convergence and there are many parameters, such as the learning rates, which need to be tuned by trial-and-error. These are both areas for further work. However, the ability of the ADAC to converge satisfactorily from scratch far outweighs these limitations.

The control performance of the ADAC is most encouraging and clearly demonstrates the feasibility of successfully using reinforcement learning for process control applications.

6.0 References

[1] Anderson, C. W., "Learning and problem solving with mulitlayer connectionist systems", *Doctoral Dissertation*, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, 1986.

[2] Anderson, C. W., "Strategy learning with multilayer connectionist representations", *Technical Report TR87-507.3*, GTE Laboratories, Wlatham, MA, 1987. Revision of the *Fourth International Workshop on Machine Learning,* pp. 103-114, June, 1987.

[3] Anderson, C. W., "Learning to Control an Inverted Pendulum Using Neural Networks", *IEEE Control Systems Magazine*, Vol. 9, pp.31-37, April 1989.

[4] Antsaklis, P., "Final Report of task Force in Intelligent Control", Technical Committee on Intelligent Control, IEEE Control Systems Society, 1993.

[5] Antsaklis, P., "Intelligent Control", in Encyclopedia of Electrical and Electronics Engineering, John Wiley &Sons, Inc., 1997.

[6] Barto A. G., Sutton, R. S., Anderson, C. W., "Neuronlike elements that can solve difficult learning control problems", *IEEE Transactions on System, Man and Cybernatics*, Vol.13, pp. 835-846, 1983.

[7] Bellman, R. E., *Dynamic Programming*, Princeton University Press, 1957.

[8] Chan, K. H., Jiang, L., Tiloston, P., and Wu, Q. H., "Reinforcement Learning for the Control of Large-Scale Power Systems", *Proceedings of 2nd International Symposium Engineering of Intelligent Systems* (EIS'2000), Paisley, UK, 2000.

[9] 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, 1990a.

[10] Chen, S., Cowan, C. F. N., Billings, S. A., and Grant P. M., "Parallel recursive prediction error algorithm for training layered networks", *International Journal of Control*, vol. 51, no. 6, pp. 1215 - 1228, 1990b.

[11] Connell, M., and Utgoff, P., "Learning to control a dynamic physical system", in *Proceedings AAAI-87*, Vol. 2, 456-460, American Association for Artificial Intelligence, Seattle, 1987.

[12] Ernst, D., Glavic, M., and Wehenkel, L., "Power system stability control: Reinforcement learning framework", accepted for publication in *IEEE Transaction on Power Systems*, 2003.

[13] Goodwin, G. C., and Paywe, R. L., "*Dynamic System Identification: Experiment Design and Data Analysis*", Academic Press, New York, 1977.

[14] Govindhasamy, J. J., McLoone, S. F., Irwin, G. W., Doyle, R. P., and French, J. J., "Neural Modelling, Control And Optimisation Of An Industrial Grinding Process", Accepted for publication in Control Engineering Practice, 2003.

[15] Guez, A. and Selinsky, J., “A trainable neuromorphic controller”, *Journal of Robotic System,* Vol. 5, No. 4, 363-388, 1988.

[16] Guez, A. and Selinsky, J., “A neuromorphic controller with a human teacher”, in *IEEE International Conference on Neural Networks,* Vol. 2, 595-602, 1988.

[17] Gunnarsson, S., “On Covariance Modification and Regularization in Recursive Least Square Identification”, 10th IFAC Symposium on System Identification, SYSID 94, pp. 661-666, 1994.

[18] Gunnarsson, S., “Combining Tracking and Regularization in Recursive Least Square Identification”, *Proceedings of the 35th IEEE Conference on Decision and Control*, pp. 2551-2552, 1996.

[19] Gupta, M. M., and Rao, D. H., “Neuro-Control Systems: A Tutorial”, In Gupta, M. M., and Rao, D. H. (Eds.), NeuroControl Systems: Theory and Application, IEEE Press, pp. 1 - 43, 1994.

[20] Handelman, D. and Lane, S., “Fast sensorimotor skill acquisition based on ruled-based training of neural networks”, in *Neural Networks in Robotics,* Bekey, G. and Goldberg K., eds., Kluwer Academic Publishers, Boston, 1991.

[21] Hoskins, J.C., and Himmelblau, D.M., “Process control via artificial neural networks and reinforcement learning”, *Computers & Chemical Engineering*, vol. 16, no. 4, pp. 241-251, 1992.

[22] Hougen, D., “Use of an eligibility trace to self-organize output”, in *Science of Artificial Neural Networks II,* Ruck, D. eds., SPIE 1966, 436-447, 1993.

[23] Hunt, K. J., Sbarbaro-Hofer, D., Zbikowski, R., and Gawthrop, P. J., “Neural Networks for Control Systems - A Survey”, *Automatica*, Vol. 28, pp. 1083-1112, 1992.

[24] Irwin, G. W., O’Reilly, P., Ligthbody, G., Brown, M., and Swidenbank, E., “Electrical power and chemical process applications”, In Neural network applications in control, Irwin, G. W., Warwick, K. and Hunt, K. J., eds., *IEE Control Engineering Series 53*, The Institution of Electrical Engineers, London, UK,1995.

[25] Iyer, M.S.; Wunsch, D.C., II, “Dynamic re-optimization of a fed-batch fermentor using adaptive critic designs”, *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1433-1444, 2001.

[26] Johansson, R., “*System Modelling Identification*”, Prentice Hall Information and System Science Series, New Jersey, 1993.

[27] Lee, C. W., and Shin, Y. C., “Intelligent Modelling and Control of Computer Hard Disk Grinding Processes”, *Proceedings of the 3rd International Conference on Intelligent Processing and Manufacturing of Materials*, pp. 829-838, 2001.

[28] Lightbody, G., “Identification and control using neural networks’, PhD dissertation, The Intelligent Systems and Control Group, The Queen’s University of Belfast, Northern Ireland, UK, 1993.

[29] Liu, D., Xiong, X., and Zhang, Y., “Action-Dependant Adaptive Critic Designs”, *Proc. of the INNS-IEEE International Joint Conference on Neural Networks,* pp. 990-995, July, 2001.

[30] Ljung, L., “*System Identification: Theory for the user*”, Prentice Halls, Englewood Cliffs, New Jersey, 1987.

[31] Ljung, L. and Söderström, T., “*Theory and Practice of Recursive Identification*”, MIT, Cambridge, MA, 1983.

[32] McLoone, S. F., “Neural Network Identification: A survey of gradient based methods”, *IEE Colloquium Optimization in Control: Methods and Applications*, London, Digest 98/521, November 1998.

[33] Michie, D. and Chambers, R., “Boxes: An experiment in adaptive control”, in *Machine Intelligence,* Dale, E. and Michie, D., eds., Oliver and Boyd, Edinburgh, 1968.

[34] Ngia, L. S. H., Sjöberg, J. and Viberg, M., “Adaptive Neural Networks Filter Using a Recursive Levenberg-Marquardt Search Direction”, *Proceedings of the 3rd Asilomar Conference on Signals, System and Computer*s, pp. 697-701, November 1998.

[35] Ngia L.S.H. and Sjöberg J., “Efficient Training of Neural Nets for Nonlinear Adaptive Filtering using a Recursive Levenberg- Marquardt Algorithm”, *IEEE Transactions on Signal Processing*, vol. 48, no. 7, pp. 1915-1926, July 2000.

[36] Park, J. W., Harley, R.G., and Venayagamoorthy, G.K., “Adaptive Critic Designs and their Implementations on Different Neural Network Architectures”, *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 1879-1884, 2003.

[37] Park, J. W., Harley, R.G., and Venayagamoorthy, G.K., “Adaptive critic based optimal neurocontrol for synchronous generator in power system using MLP/RBF neural networks”, *Conference Record - IAS Annual Meeting (IEEE Industry Applications Society)*, vol. 2, pp. 1447-1454, 2002.

[38] Passino, K., “Bridging the Gap Between Conventional and Intelligent Control”, IEEE CSM, pp. 12-18, June, 1993.

[39] Prokhorov, D., Santiago, R., and D. Wunsch, “Adaptive Critic Designs: A Case Study For Neurocontrol”, *Neural Networks*, vol. 8, no. 9, pp. 1367-1372, 1995.

[40] Prokhorov, D., and Wunsch, D., “Adaptive critic designs,” *IEEE Transactions on Neural networks,* Vol. 8, pp. 997–1007, Sept. 1997.

[41] Radhakant, P., and Balakrishnan, S. N., “Proper orthogonal decomposition based optimal neurocontrol synthesis of a chemical reactor process using approximate dynamic programming”, *Neural Network*, vol. 16, pp. 719-728, 2003.

[42] Riedmiller, M., “Concepts and Facilities of a neural reinforcement learning control architecture for technical process control”, *Neural Computation and Application Journal*, vol. 8, pp. 323-338, Springer Verlag London, 1999.

[43] Rovnak, J. A. and Corlis, R., “Dynamic matrix based control of fossil power plants”, *IEEE Transactions on Energy Conversion*, vol. 6, no. 2, pp. 320-326, 1991.

[44] Salgado, M. E., Goodwin, G. C., and Middleton, R. H., “Modified Least Squares Algorithm Incorporating Exponential Resetting and Forgetting.”, *International Journal of Control*, vol. 47, no. 2, pp. 477-491,1988.

[45] Sammut, C., “Experimental results from an evaluation of algorithms that learn to control dynamic systems”, in *Proceedings of the Fifth International Conference on Machine Learning,* 437-443, Morgan Kaufman, San Mateo, California, 1988.

[46] Sofge, D. A., and White D. A., “Neural network based process optimization and control”, *Proceedings of the IEEE Conference on Decision and Control*, vol. 6, pp. 3270-3276, 1990.

[47] 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, March 2001.

[48] Sutton R. S., “Learning to Predict by the Method of Temporal Differences”, *Machine Learning*, Vol. 3, pp. 9-44, 1988.

[49] Sutton R.S., “Implementation Details of the TD(λ) Procedure for the Case of Vector Predictions and Backpropagation”, *GTE Laboratories Technical Note TN87-509.1*, Aug., 1989.

[50] Sutton, R.S., and Whitehead, S.D., “Online learning with random representations”, *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314-321, 1993.

[51] Venayagamoorthy, G.K., Harley, R.G., and Wunsch, D.C., “A nonlinear voltage controller with derivative adaptive critics for multimachine power systems”, *IEEE Power Industry Computer Applications Conference*, p 324-329, 2001.

[52] Venayagamoorthy, G.K., Harley, R.G., and Wunsch, D.C., “Excitation and turbine neurocontrol with derivative adaptive critics of multiple generators on the power grid”, *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 984-989, 2001.

[53] Venayagamoorthy, G.K.; Harley, R.G.; Wunsch, D.C., “Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator”, *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 764 -773, 2002.

[54] Werbos, P. J., “Consistency of HDP Applied to a Simple Reinforcement Learning Problem”, *Neural Networks*, vol. 3, no. 2, pp. 179-189, 1990.

[55] Werbos, P. J., “A Menu of Designs for Reinforcement Learning Over Time”, in Miller, W. T., Sutton, R. S. and Werbos P. J. eds, *Neural Networks for Control*, MIT Press, Cambridge, MA, pp. 67 - 95, 1990.

[56] Werbos, P. J., “Approximate Dynamic Programming for Real-Time Control and Neural Modelling”, in White, D. A. and Sofge, D. A. eds, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, New York, pp. 493 - 525, 1992.

[57] White, D. A., and Sofge, D. A., “Neural network based control for composite manufacturing”, American Society of Mechanical Engineers, Materials Division (Publication) MD, *Intelligent Processing of Materials*, vol. 21, pp. 89-97, 1990.

[58] Widrow, B., “The original adaptive neural net broom-balancer”, in *International Symposium on Circuits and Systems, 351-357, 1987.*

[59] Widrow, B., Gupta, N. and Maitra, S., “Punish/Reward: Learning with a Critic in Adaptive Threshold Systems”, *IEEE Transactions on Systems, Man and Cybernatics*, vol. 3, No. 5, pp. 445 - 465, 1973.

[60] Willis, M. J., Di Massimo, C. D., Montague, G. A., Tham, M. T., and Morris, A. J., “Artificial neural networks in process engineering”, IEE Proceedings - Part D: Control Theory and Applications, Vol. 138, No. 3, pp. 256-266, 1991.

[61] Wu, Q. H., and Pugh, A. C., “Reinforcement learning control of unknown dynamic systems”, *Proceedings of the IEE, Part D: Control Theory and Applications*, Vol. 140, pp. 313-322, 1993.

[62] Wu, Q. H., “Reinforcement learning control using interconnected learning automata”, *International Journal of Control*, vol. 62 (1), pp. 1-16, 1995.

[63] Young, P.C., “*Recursive Estimation and Time Series Analysis*”, Springer-Verlag, Berlin, 1984.

[64] Zeng, X., Zhou, J., and Vasseur, C., “A strategy for controlling nonlinear systems using a learning automaton”, *Automatica*, vol. 36, pp. 1517-1524, 2000.