# REINFORCEMENT LEARNING IN CONTROL

**M. Sheppard[1], A. Oswald[2], C. Valenzuela, G. Sullivan and R. Sotudeh**

**University of Teesside, Middlesbrough, Cleveland, TS1 3BA, UK**

## ABSTRACT

During its melt cycle, an arc furnace causes disturbances of the electrical supply. Existing measurement techniques for this application lead to corrective rather than predictive compensation. The use of neural networks to control the compensation is being considered, in particular reinforcement learning strategies which require no pre-training and which can adapt to a dynamically changing environment.

Several reinforcement learning techniques have been considered by examining their effectiveness in learning to balance a pole on a moving cart without prior training. The network is required to produce an appropriate control action in response to the current world state in order to maintain the pole and cart position within acceptable limits. One reason for investigating this is the belief that many of the characteristics of the pole-balancer are analogous to the problem of compensating for reactive power disturbances in the arc furnace.

This paper presents a comparative review of these reinforcement learning strategies.

## KEYWORDS

reinforcement learning, reactive power compensation, arc furnace, associative search, Q-learning, stochastic real-valued learning

## INTRODUCTION

The power transmission network is shared by both domestic and industrial users. However some industrial installations such as the electric arc furnace can cause disturbances to this network; the principle of the arc furnace is to short circuit the electrical supply which heats the metal in the furnace, causing it to melt. The resistance of the metal continually changes as it progresses from solid to molten state. This rapidly changing resistance gives rise to voltage dips at the point of common coupling to the supply which are passed to the domestic users. These voltage dips can affect sensitive electronic equipment and generate annoying flicker on tungsten filament lamps. Compensation for the changing resistance can be performed by adding a capacitive current to the system but determining this level of capacitance is difficult. Compensation algorithms exist which consider a window of previous electrical spectral values and as a result estimate the required compensation. Through their reactive nature, the algorithms sometimes produce compensation which is inappropriate to a rapidly changing world. Also, if shown the same situation repeatedly, they will always generate the same level of compensation, regardless of its effectiveness. A more desirable mechanism would be one that could be predictive and that could learn from previous experience. It is hoped that neural networks could be of use in producing a better control strategy.

[1]mark@teesside.ac.uk
[2]oswald@teesside.ac.uk

A controller to balance a pole upright on a movable cart is an isomorph of the problem of compensating for the arc furnace disturbance. In the pole world the system is required to examine the parameters describing the world and produce an appropriate control action to minimise the angle of the pole by pushing the cart left or right. In the arc furnace problem, a controller also examines the world state and is then required to produce a level of compensation which will minimise the voltage disturbance. This paper outlines the investigations into neural strategies for the control of the pole balancer isomorph and shows how the findings of this work have been used in an initial implementation of an arc furnace compensator.

## THE POLE BALANCING WORLD

Figure 1 shows a hinged pole mounted on a wheeled cart. The cart can move left or right along a rail, starting from a central point. In order to balance the pole, the system must apply force to the cart in either direction to prevent the pole from falling past an agreed failure angle, whilst keeping the cart within an agreed distance from the central point on the track.

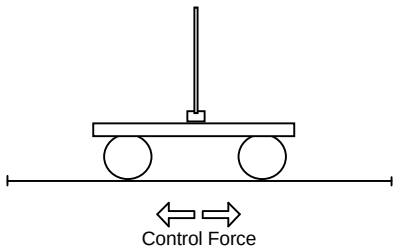


Fig. 1 The Pole/Cart System

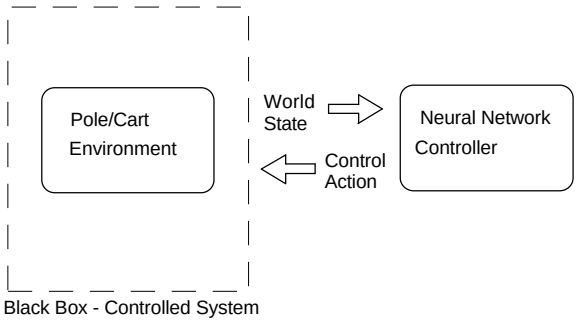


Fig. 2 Black box approach to controlled system

Since the pole balancer is an isomorph of the arc furnace problem the thinking behind the development of the neural controller is to treat the system it controls as a black box (See fig. 2). This allows the replacement of the pole environment with the arc furnace or any other application areas which are, at least, homomorphs of the pole problem. Consequently communication between the controlled system and its controller is kept to a minimum, world parameters are passed into the controller, and a resulting control signal is returned. The behaviour of the black box pole world has been modelled by two differential equations, taken from Barto *et al* (1983). These have been implemented using a high speed, high order Runge-Kutta method developed by Dormand and Prince (1989).

## NEURAL LEARNING

Probably the most commonly used neural learning strategy is supervised learning. A network is taught through repeated presentations to learn and to generalise from a given training set. Middleton (1992) and later Dixon (1993) have looked at the application of supervised networks to the arc furnace problem. These networks were trained on input/output mappings collected from an arc furnace simulator which measured compensation based upon static algorithms developed by Etminan and Sotudeh (1987). These algorithms estimate the level of reactive power present in the system and from this compensation can be determined. In both cases the network learned to mimic the training and performed nearly as well as the algorithms. However, since the algorithms producing the training data were not predictive, nor able to learn from their mistakes, a supervised back-propagation network could never meet these criteria either. If a neural network is to perform better than the algorithmic approach, then an alternative learning strategy was needed.

Reinforcement learning (RL) describes a learning paradigm where a system attempts to improve its behaviour based on the results of previous actions (Sutton, 1992). These results are returned from the outside world as a level of "reward". Each time the network controller is called upon to make a control decision it chooses an action which is expected to yield the highest reward based on its previous findings. In the arc furnace world compensation which resulted in no voltage drop could be seen as the optimal performance and so would yield the highest reward. Increasingly less effective compensation would increase the voltage drop, and the level of reward should decrease. Mapping this idea to the pole world, the pole angle replaces the voltage drop, and compensation producing an upright pole would yield maximum reward. Therefore in both problems we know what we want the network to achieve and we reward it based on its level of achievement, but the task of determining how best to achieve it is left to the neural controller to learn.

There are many strategies based on reinforcement learning, but they all follow the same generic algorithm shown in fig. 3. It must be stressed that reinforcement learning is on-line adaptive learning; it does not have implicit training and recall stages as with supervised learning, but instead interleaves control with the modification of internal weights.

```
repeat
        receive parameters describing the current world state
        generate a control action for this state, based on internal weights
        execute the action on the world
        receive reward for the previous action
        if favourable reward then
                alter weights so as to promote this action in similar
                situations in the future
        else
                alter weights so as to discourage this action in similar
                situations in the future
forever
```

Fig. 3 A Generic Reinforcement Learning Algorithm

## POLE BALANCING CONTROLLER IMPLEMENTATIONS

The pole balancer analogy has been implemented using three different reinforcement learning strategies. The first, Associative Search, is included as a mechanism which requires minimal feedback, namely a single signal on failure, to meet the control objective. Q-Learning is based on Markovian dynamic programming and Stochastic Real-Valued learning has the property of operating on a continuous world.

### Associative Search Learning

An early model of reinforcement learning was presented by Barto, Sutton and Anderson (1983). The ASE, or Associative Search Element takes a representation of the environment and learns to associate successful outputs to these inputs after repeated learning trials. The original paper demonstrated that an RL system could learn to balance a pole within given limits.

The system relies on a simple feedback signal from the world. When the pole falls over, or more precisely moves outside the angular or cart distance bounds, the environment supplies an error signal, to the controller. On the basis of this signal, the system learns to avoid failure. The dominant factor in reinforcement learning systems is the so-called temporal credit question, i.e. "what was responsible for the

current situation". In this case the question becomes "which control decisions were responsible for the pole's falling over?". As only one failure signal arrives, the controller needs to apportion blame to the moves leading up to the eventual failure. The final decision is ultimately at fault, the decision before that was nearly as responsible, and so on back through the control decisions. ASE learning implements this reasoning with a decaying eligibility trace, i.e. a responsibility which decays as the time from each specific decision increases.

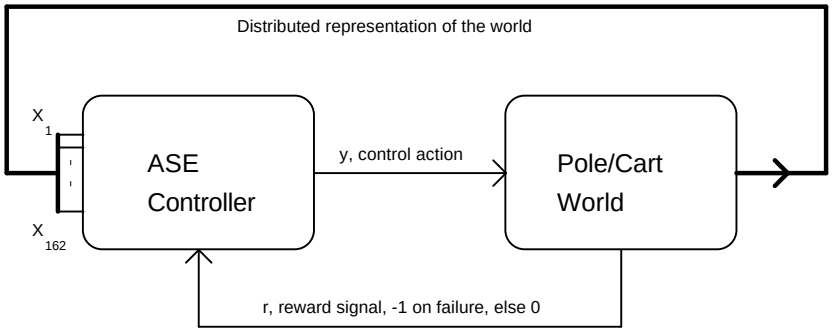


Fig. 4 ASE Controller

Figure 4 shows the ASE controller supplying the pole balancing world with control actions in response to the current world position. In return the world supplies the controller with the new world state and the cycle repeats. If however the pole should fall beyond the predefined limits, the environmental parameters would be reset to zero, i.e. an upright stationary pole on a non-moving cart in the middle of the track, and then the system would modify its weights in the light of the recent failure.

The world state is represented as three ranges for the cart position, three for the cart velocity, six for the pole angle and three for the angular velocity resulting in 162 (3 x 3 x 6 x 3) possible situations to which the controller is expected to react (See Michie and Chambers, 1966). The world state is presented as 162 input lines, all set to zero except the line representing the current world state. For example if the cart position fell into range one, the cart velocity into range two, the angle and angular velocity into ranges four and one respectively, the combination would set input line 115 to 1, and clear all of the other inputs to 0. The control action is derived from the sign of $y$, a negative sign causing the cart to be pushed right, otherwise moving the cart left.

$$y(t) = f\left( \sum_{i=0}^{162} w_i(t) x_i(t) \right) \quad (1)$$

The result of this action will be reflected in the new state presented to the controller at time $t+1$. If the pole falls outside the permitted bounds, the weights will be modified subject to multiplication by a learning rate constant, $\alpha$, set in these trials to be 0.95 and the failure signal, $r$, set to -1. Equation 2 shows how the new weights are calculated.

$$w_i(t+1) = w_i(t) + \alpha r(t) e_i(t) \quad (2)$$

The term $e$ in the equation refers to eligibility, the ASE method of solving the temporal credit problem. Each input line not only has a weight on its input line, but also an eligibility value. This value is set to one when the input line is used, i.e. when that state is entered. At each time step, all eligibility weights are multiplied by a decay factor, $\delta$, in our trials set to 0.95. From this weight the blame for any failure can be apportioned to the states leading to failure. The eligibility at time $t+1$ is now calculated:

$$e_i(t+1) = \delta e_i(t) + (1-\delta) y(t) x_i(t) \quad (3)$$

## ASE Implementation Results

The first implementation of the ASE pole balancer was as developed in Barto *et al*. Initial weights were all set at 0.5, and limits for the pole angle, $\theta$, and cart distance from centre, $x$, set at 12 degrees and 2.4 units respectively. Figure 5 shows the number of time steps to failure on each trial. After 20 trials the system reached a stable state balancing the pole for in excess of three thousand steps. It must be noted that the levels are well within the limits given above.

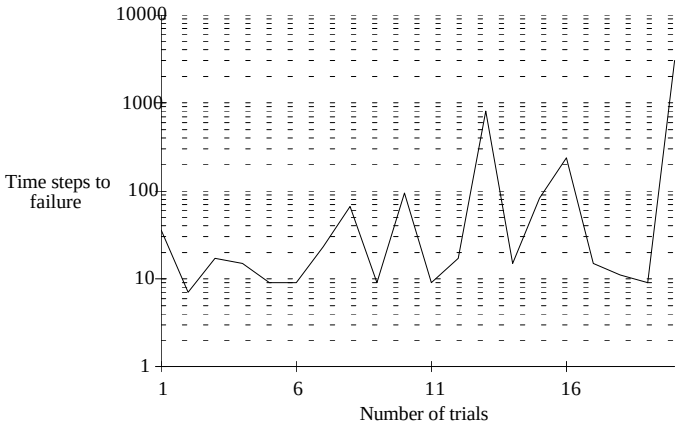


Fig. 5 ASE learning over time

The $x$ value settled at approximately 0.08 units. In an attempt to improve this, the initial starting weights of the system were kept at 0.5. but with a random sign. When there was a distribution of positive and negative starting weights, the $x$ value settled between 0.015 and -0.025 in trials.

One further extension to the ASE system was an attempt at shaping. Shaping is a psychological term whereby animals are trained to tackle increasingly harder problems, building on the skill they have acquired. In the pole balancer the failure limits were closed in after successfully maintaining the pole within the limits for 1000 steps. However the system was unable to balance the pole with any more accuracy than +/- 2 degrees from upright. Reasons for this could be the granularity of the control action, set at +/- 10 Newtons or the number of boxes used.

The results of the ASE network showed that a reinforcement learning system can learn to control an unknown system. The simplicity of the reward mechanism makes it attractive as a general control system.

## Q-Learning

Watkins linked reinforcement learning methods and dynamic programming (Watkins, 1992; Whitehead and Ballard, 1991). Presenting RL as a dynamic programming problem, the aim of a reinforcement learning controller is to maximise the reward it receives at successive states through the state space. In a state *s* at time *t* the agent can perform various actions which will take it to new states. The mechanism by which it decides which action to perform is the policy function, and the learning mechanism should move this policy toward the optimum policy, i.e. at every decision point, to select the action which will result in the highest future reward. The hybrid method of RL and dynamic programming proposed was Q-Learning.

This policy is called $Q$, or the action value function. Given a state and an action, $Q$ will give the expected return. Return can be regarded as an immediate reward for a state, plus an estimate of future reward that can be achieved from that state. In chess, a move which allows you to take your opponent's Queen may have a high immediate reward, but then allows your King to be check-mated has a very low long term reward. Return *r* at time *t* is defined as:

$$r_t = \sum_{n=1}^{\infty} \gamma^{n-1} r_{t+n} \tag{4}$$

where $\gamma$ is the discount factor, typically 0.95. A function $V$, the value function can be defined, which will give the expected return for a state *x* at time *t*. Looking at the equation for return (4), as *n* becomes very large, $\gamma^{n-1}$ becomes very small and the rest of the return will be negligible. Therefore Watkins used a method called "corrected n-step truncated return" which calculates the return up to time *t+n* and then for the remainder, finds the value of $V$, which will give future expected return, for the state at time *t+n*. Therefore when given *n*, the number of time steps to be calculated, the corrected n-step truncated return can be found as:

$$r_t^{(n)} = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} + \gamma^{n} V_t(x_{t+n}) \tag{5}$$

Initially the system will have no information as to which action it should perform at each state. In this implementation, the value of $Q$, the action value function, for each action and state pair will be random. Once the action has been executed, the real reward will be known, and the error between the predicted return and the actual return can be found, giving the $Q$ value at the next time step.

$$Q_{t+n}(x_t, a_t) = Q_t(x_t, a_t) + \alpha \left( r_t^{(n)} - Q(x_t, a_t) \right) \tag{6}$$

Watkins used 1-step Q-Learning, in which the action value function $Q$ is updated after a delay of only one step, i.e. as soon as the actual return for the next state is known.

Figure 6 shows an algorithm for Q-Learning.

```
repeat
        receive current world state
        if random action required then
                generate a random control action
        else
                generate a control action based on the policy for the current state
        execute the action on the world
        receive reward from the previous action
        calculate error between actual and expected reward
        alter action value function in light of error
        amend policy to ensure it still dictates action yielding maximum reward
forever
```

Fig. 6 Algorithm for Q-Learning

## Q-Learning Implementation Results

The Barto scenario was duplicated as much as possible using 1-step look-ahead Q-Learning rather than ASE learning. Q-Learning requires a reward value to be returned from the environment after each control action and the strategy adopted used the value of (1/angular velocity) as a reward. It was hoped that this mechanism would favour a more stable system. However the results were poor and the pole was never successfully balanced for more than 940 time steps in the repeated trials undertaken. Several alternative reward strategies were used, including reward based on improvement since the last control action, but none

of these resulted in a significant improvement. Figure 7 plots 1000 trials of Q-Learning showing the number of time steps to failure on each.

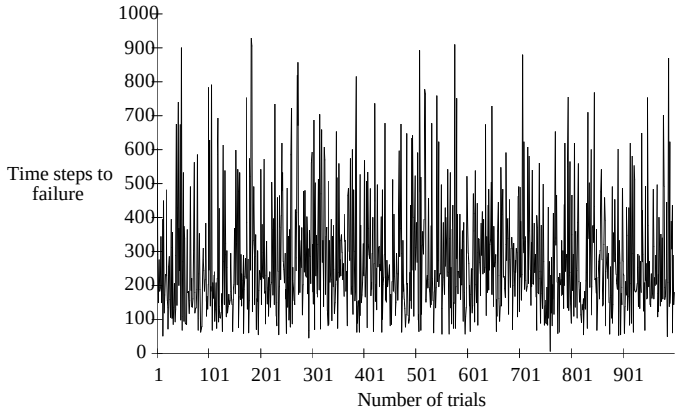


Fig. 7 Q-learning over time

## Stochastic Real Valued Learning

The previous learning strategies have relied on an environment split into discrete boxes. The granularity of the box quantitisation determines the granularity of control. If a fine level of control is required, the boxes are made smaller, which considerably increases the number of states. As the number of states becomes much larger, the system suffers as the computational overhead increases.

.

An alternative is the Stochastic Real Valued (SRV) approach proposed by Gullapalli (1990, 1992) which takes real values of variables directly from the environment and uses these to determine the required control action. The method does not suffer from heavy computational burdens and permits continuous values for the control action.

The SRV approach attempts to determine the optimal control action *a* for the world state *x* presented to it at time *t*. Using random search based on a normal distribution to estimate the optimal action. Depending on the success, the mean of the normal distribution is moved toward actions which produce higher rewards *r(t)*, and the standard deviation is made smaller as the system begins to focus around the "correct" action.

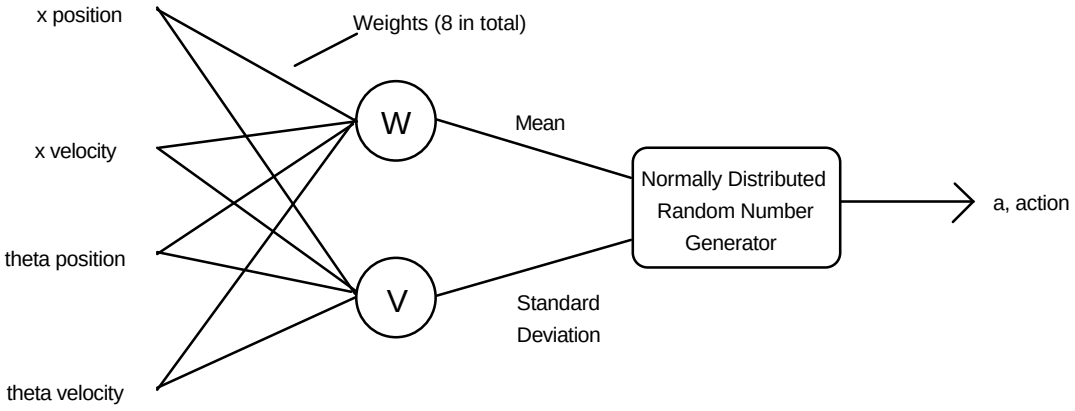


Fig. 8 An SRV unit constructed from two neural nodes

Figure 8 shows how an SRV unit consists of two neural nodes in parallel; the w-node attempts to learn the desired control action, and the v-node, controls the amount of search. As with the previous learning methods, the system is shown four parameters from the pole balancing environment, the cart position, velocity, angle and angular speed. In response the controller produces a control action, which is applied to

the world , resulting in a new world state and a measure of performance. With four environmental variables fully interconnected to two SRV nodes, there are eight weights in total, four leading to the w-node, controlling the mean, and four to the v-node controlling the standard deviation.

The mean for the normal distribution is calculated from:

$$\mu(t) = \sum_{i=1}^{n} w_i(t)\, x_i(t) + w_{thres}(t) \tag{7}$$

The other node indirectly produces the standard deviation. In the early stages of learning, one would hope the system would explore, searching for the optimal control action. Before the optimal action is found, the reward associated with the sub-optimal actions is lower than could be expected with optimal actions. The *v* node attempts to learn the reward associated with each control action. As the error between the expected reward and real reward decreases, the standard deviation should be lowered, thus limiting the search space. The expected reinforcement is given as:

$$\hat{r}(t) = \sum_{i=1}^{n} v_i(t)\, x_i(t) + v_{thres} \tag{8}$$

From this, the standard deviation can be found. When the error in reward estimation is zero, the system has found the optimal action for the given world state, and no further search is required. This is simply achieved by forcing the standard deviation to be zero.

$$\sigma(t) = \max\left( \left(\frac{1.0 - \hat{r}(t)}{5.0}\right), 0.0 \right) \tag{9}$$

Given the mean and standard deviation a normally distributed random number generator produces a real control action.

$$a(t) = \psi\left( \mu(t), \sigma(t) \right) \tag{10}$$

Like Q-Learning, SRV interleaves control with learning, using a two stage algorithm; the first half generating a control action; the second modifying the internal weights based on the reward received from the environment. Since there is a set of weights leading to each of the neural nodes, two sets of weight modification equations are required. The algorithms are based on the gradient descent methods found in back error propagation, but the magnitude of the error is implied by the difference between expected and actual reinforcement.

$$\Delta_w(t) = \left( r(t) - \hat{r}(t) \right)\left( \frac{a(t) - \mu(t)}{\sigma(t)} \right) \tag{11}$$

The weights and threshold are then moved in the direction of the correct solution, multiplied by a learning rate parameter.

$$w_i(t+1) = w_i(t) + \alpha \Delta_w(t)\, x_i(t) \tag{12}$$

$$w_{thres}(t+1) = w_{thres}(t) + \alpha \Delta_w(t) \tag{13}$$

The v weights which are attempting to predict the expected reinforcement to aid in calculating the standard deviation are trained in a similar way. The error is known here; it is the difference between expected and actual reward.

$$\Delta_v(t) = r(t) - \hat{r}(t) \tag{14}$$

Again the weights are modified to counteract this error, subject to a learning rate parameter.

$$v_i(t+1) = v_i(t) + \beta\Delta_v(t)\, x_i(t) \qquad (15)$$

## SRV Implementation Results

By far the most successful of the three implementations was the Stochastic Real-Valued learning. The improvement reward mechanism tried with Q-Learning was used, rewarding the system if it decreased the pole angle, or moved the cart closer to the central point. The initial runs balanced the pole for as many as 100,000 time steps. However the behaviour when in this so-called balanced state was not stable, and system exhibited a small degree of drift in the cart position, leading to eventual failure.

The drift was examined and a solution found. The system was continually searching for an optimal policy and when very good behaviour was exhibited the returns from this search were found to be counter productive, i.e. it was trying to be "too clever". A graduated scale of search was overlaid onto the mechanism calculating standard deviation. If the system was within 5% of the given angular limit, the standard deviation would be set to zero irrespective of the value calculated by the weights. As a result of this intervention policy the system learned to balance the pole for 2 million time steps, at which time the trial was stopped. Figure 9 shows the learning trials in the improved system and fig. 10 shows a sample of 500 time steps from the balanced run illustrating how the pole angle was maintained within +/- 0.015 degrees of upright.

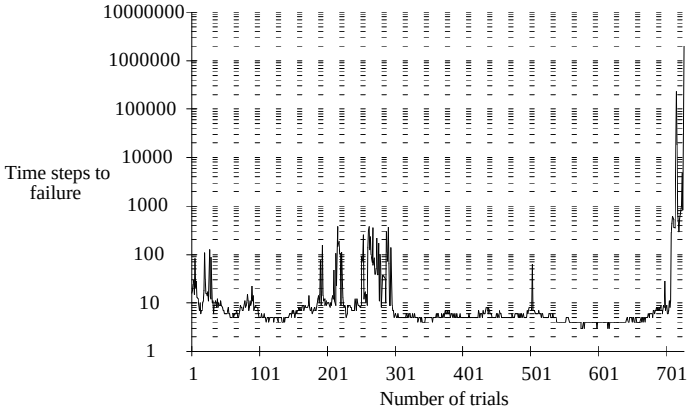


Fig. 9 SRV learning over time

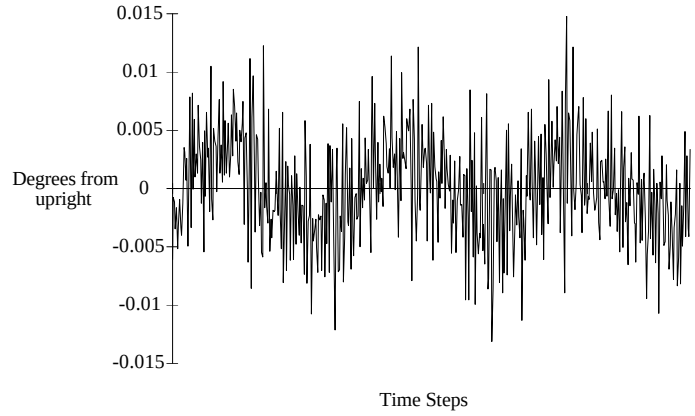


Fig. 10 Pole angle whilst in balanced state under SRV control

## ARC FURNACE CONTROLLER IMPLEMENTATION

Since the Stochastic Real-Valued learning strategy performed significantly better than the other learning methods in the pole balancing problem, it was chosen for the initial implementation of the arc furnace controller. The black box neural network controller was linked to an arc furnace simulator. This piece of software written by Sina Etminan (1990) mimics the disturbances generated by a furnace during its melt cycle. The software was used during the development of the existing compensation algorithms.

In the implementation, four parameters were passed from the arc furnace simulator to the controller, the voltage at the point of common coupling (p.c.c.), the supply current, and the rates of change of each of these values. The neural net estimates the level of compensation required, and the simulator returns a reward level, indicating the success of the compensation. In the early trials, the reward decreased as the RMS voltage at the p.c.c. increased.

Initial results show that the neural controller can learn to maintain the voltage within +/- 1% of the desired level. Further enhancements to the system are now being developed to improve on this.

## REFERENCES


Barto, A. G., R. S. Sutton and C. W. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13, 834-846.

Bellman, R. E. (1957). *Dynamic programming*, Princeton University Press.

Dormand, J. R. and P. J. Prince (1989). Practical Runge-Kutta Processes. *SIAM Journal of Scientific and Statistical Computing*, 10, 5, 977-989.

Dixon, C. (1993). *Neural Net for the Compensation of Electrical Supply Disturbances.* University of Teesside.

Etminan, S. (1990) *Simulation of High-Speed Reactive Power Compensations for Power System Disturbances*, Sunderland Polytechnic, 1990.

Etminan, S. and R. Sotudeh (1987). *Fast Converging Reactive Power Measurement Techniques for High Speed Static VAr Compensations*. Sunderland Polytechnic.

Gullapalli, V. (1990). A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions, *Neural Networks*, 3, 6, 671-92.

Gullapalli, V. (1992). *Reinforcement Learning and its Application to Control*, University of Massachusetts.

Middleton, R. (1992). *Neural Net for the Compensation of Electrical Supply Disturbances.* University of Teesside.

Michie, D. and R.A. Chambers (1968). Boxes: An Experiment in Adaptive Control. In: *Machine Intelligence* (D. Michie, ed.) Vol. 2. Edinburgh University Press.

Sutton, R.S. (1992). The Challenge of Reinforcement Learning, *Machine Learning*, 8, 3-4, 226-255.

Watkins, C. J. C. H. (1992). Technical Note: Q-Learning, *Machine Learning*, 8, 3-4, 279-292.

Whitehead, S. and D. Ballard (1991) Learning to Perceive and Act by Trial and Error, *Machine Learning*, 7, 45-83.