Modular SRV Reinforcement Learning Architectures for Non-linear Control

V.Paraskevopoulos[1]

M.I.Heywood[2]

C.R.Chatwin[1]

[1]University of Sussex, iims, School of Engineering, Falmer Brighton, BN1 9QT, U.K.

{V.Paraskevopoulos, c.r.chatwin}@sussex.ac.uk

Dokuz Eylül University, Dept. of Computer Engineering, Bornova, Izimr 35100, Turkey

mheywood@cs.deu.edu.tr

Abstract

This paper demonstrates the advantages of using a hybrid reinforcement–modular neural network architecture for non-linear control. Specifically, the method of ACTION-CRITIC reinforcement learning, modular neural networks, competitive learning and stochastic updating are combined. This provides an architecture able to both support temporal difference learning and probabilistic partitioning of the input space. The latter is formed with the aid of competitive learning algorithms, so as to ensure suitable partitioning of the experts in the modular network.

Application of this methodology to the pole-balancing benchmark non-linear control problem demonstrates superior partitioning of the input space, bettering that of equivalent reinforcement networks; whilst avoiding the learning-to-learn nothing effect, as is often the case when performing gradient decent over problems requiring adaptation over long temporal dependencies.

Keywords : Modular neural networks, reinforcement learning, non-linear control.

1. Introduction

The non-linear modeling capability of neural networks has frequently provided the rationale for investigating the appropriateness of the technique to applications in non-linear control. The control problem, in its most general setting, represents the combination of a predictor and a function approximator, although in most contexts sufficient data is available to enable the problem to be modeled as a function approximator alone. By using the more general context the predictor is attempting to ensure that actions taken at some present time, *t*, by the approximator, do not result in an undesirable action at some future point in time $t + n$. A reinforcement neural network architecture naturally supports the predictor–approximator combination. That is to say the error function has little distance and/ or state information, where this may also be subject to temporal discounting. Architectures initially designed for the reinforcement problem where limited to binary responses [1, 2]. Gullapalli among others generalized the architecture to support real valued responses [3]. Furthermore, the Stochastic Real-Valued (SRV) architecture proposed by Gullapalli naturally encompasses the division of duties between predictor and function approximator using unique networks. This is important as it explicitly supports the decomposition of the training process into separate stages, an important practical consideration. The requirement for learning across temporal duration's may be addressed using the method of temporal differences [4]. For example, such a combination has recently been employed for iterative rescheduling of batch production plants [5].

Partitioning plays a central role in assisting fast convergence in reinforcement learning i.e. simplifying the task of relating states from the input space to states in the output space. The objective of this work is to provide the capability for hierarchical partitioning within the CRITIC and ACTION networks themselves. One candidate for such a requirement is the Modular Network of Jacobs and Jordan, where this enables the combination of simple networks through the concept of gated expert networks [6]. That is to say the function approximation activity is decomposed into more simple elements and then recombined.

Such a method is different from input space partitioning, where this is either defined *a priori* [2] or encompassed within the adaptive process [7]. The former method obviously has limited application, whereas the second often results in far too many clusters. This then makes the identification of explicit control rules difficult. Here, however, a scheme based on unsupervised clustering is used. The objective is not just to provide a data driven method for deriving the input space partitions, but also to avoid combinatorial increases in the number of clusters necessary to cover

the input space. The intention is therefore to enable partitioning over three levels of the architecture. That is the ACTION–CRITIC decomposition naturally supports the predictor–approximator partition. Modular networks are then employed in each, thus decomposing the approximation problem further. Finally, data driven clustering is used to form initial partitions of the input space.

Reinforcement learning, due to its temporal application base, typically requires substantive training before convergence is achieved [8]. As indicated above partitioning is used extensively throughout the architecture in order to improve the efficiency of the adaptive process. The methodology used to adapt the various components of the network, however, also has significant effects on the rate of convergence. In particular a winner-takes-all scheme is employed in the gating network of the modular networks. This restricts weight updating to specific expert networks and components of the gate at any one time.

The remainder of the paper defines how the ACTION–CRITIC architecture employs Mixture of Experts paradigm to form the partitions as part of the adaptive process. Section 2 describes the algorithm; section 3 summaries the details of the combined architecture; section 4 demonstrates the application of the system to the pole-balancing benchmark problem; section 5 summarizes related work; and section 6 concludes the paper.

2. Defining the Training Algorithms

As indicated above, the reinforcement learning context represents an optimization problem in which little distance and/ or sign information is available to guide the adaptive process. To do so the Stochastic Real-Valued (SRV) method of Gullapalli [3] is used to split the problem into two components. Specifically, the reinforcement learning methodology [1, 2, 9] uses an ACTION network for optimizing the response from the environment, r; and the CRITIC network for providing a more descriptive cost function, $\hat{r}$. In effect, the derived cost function of the CRITIC is used to control the stochastic nature of the ACTION network.

The role of the CRITIC network is therefore to predict the optimality of some previous action, $\hat{y}(t-1)$; supplied as a combination of stochastic exploration and ACTION network response, $y(t-1)$. The cycle between environment, CRITIC and ACTION network repeats until the environment supplies a payoff denoting satisfaction of some end condition. A further enhancement, however, is necessary. The reinforcement learning problem as defined above represents the case of learning within an environment with no temporal discounting. In order to support directly the learning of sequences, the

method of temporal difference learning is incorporated [4]. This reformulates the cost function such that the CRITIC network acts as a multi-step look-ahead predictor evaluating the significance of the present ACTION network response in terms of the future expected payoff [5]. The first enables the introduction of modular networks in the CRITIC and ACTION components, thus facilitating faster, more transparent learning. The second integrates identification of input space clusters using unsupervised learning within the overall training algorithm.

As there is no provision for recurrent state feedback in the network, the input space is partitioned into sectors. The network then learns to associate the various sectors with good or poor actions and predicted performance. Previous approaches either required *a priori* specification of the relevant partitions of the input space [2], or used an adaptive fuzzy systems context [7] (generally results in a large number of rules). With this work, a different approach is taken to introducing the partitions. The objective is to speed the learning process whilst minimizing the curse of dimensionally associated with partitioning the input space using grid or mesh based approaches.

In particular the Mixtures of Experts (MoE) framework is employed, one each for the CRITIC and ACTION networks; section 3.1. Empirical experimentation however, indicated that to perform weight updating across all experts within the CRITIC and ACTION elements of the architecture did not result in the required convergence. That is to say, the gate did not learn to partition the problem. This was ratified by way of a winner-takes-all paradigm in which the gate response with the highest probability indicates the 'winning' expert network. The weights of this expert and those associated with the associated gate are then updated. The remaining gate parameters and expert networks are left unchanged. This differs from previous approaches in which the experts are first trained as feature vectors and the gate network is then applied to combine the features as per the cost function [10]. In this case unsupervised clustering, applied to the input space but integrated into the overall training procedure, provides this property; section 2.2. This combination of the MoE architecture and rough clustering of the input space shifts the emphasis away from clustering the input space alone. By doing so the trained network is more opaque. Furthermore, the hierarchy of clusters enables the removal of the hidden layer in the gates and experts. This reduces the significance of the initial network conditions, as well as speeding the adaptive process, without significantly compromising the overall network capacity for performing non-linear mappings. The overall architecture of the proposed network is summarized by figure 1.

The manner in which patterns are presented to the network also has significant bearing on the network's ability to converge. The routine used in this case begins by applying patterns further to the solution state and gradually includes states requiring less difficult control actions. The aim here is to

avoid learning-to-learn nothing. The following subsections, detail the training algorithm employed to adapt the CRITIC and ACTION networks individually and the integration of the input space clustering procedure.

2.1. Training algorithm

As indicated above, the principle objective of this work is to provide an architecture, applicable to problems of a temporal nature, which maximizes the use of partitioning during adaptation. Use of such a modularized architecture enables a piecewise application of the training process, such that different modules of the network are adapted individually. That is to say, as the CRITIC and ACTION networks perform different tasks, training should focus on the needs of each individually. Furthermore, in order to assist adaptation of the network as a whole, the training algorithm operates in two phases: pre-training and training. The first phase begins with the CRITIC alone. For this phase only the CRITIC is pre-trained whilst the ACTION network is replaced by a suitable exciting function; in the case of the pole-balance problem a sinusoid is used. The objective is to enable the CRITIC network to see a suitable cross-section of plant states. That is, pre-training visits are guaranteed to visit conditions in which the plant is far from optimal, hence the network has a lot to learn. Adaptation however, ends when the plant enters the null condition, synonymous with a zero force, and the CRITIC correctly predicts this (i.e. zero weight update coinciding with zero force). This action is necessary because if training continues within the stable regions of control, then the network tends to learn-to-learn nothing[1]. Completion of this phase produces a CRITIC with some experience, hence able to pre-train the ACTION network.

In the second phase of pre-training, the pre-trained CRITIC network is used to judge the performance of the ACTION network. The aim now is to tune the latter in such a way that it manages to control the plant for ideally ***all*** training patterns. Caution is taken ***not*** to present the network with patterns implying a zero initial force. Again this would result in the learning-to-learn nothing scenario. Likewise, weight updating is inhibited when the control action approaches the nearly balanced region. Completion of this phase produces an ACTION network capable of completing the training of the CRITIC.

[1] It is acknowledged that this definition is also far from optimal. Just because the plant controller enters a zero action state, the plant is not necessarily continuously stable. However, to assume anymore than this would erode the reinforcement learning context.

The aim of the final training cycle is to fine-tune the gate network of the CRITIC modular network, with the ACTION network providing the control actions. The CRITIC is trained with the same initial conditions as before, but is now trained until the gate network weight changes are minimized. Although this phase enhances CRITIC performance, the same does not appear to be the case for the ACTION network. As a result a second training cycle of the latter is not performed.

2.2. Unsupervised clustering and stochastic parameter adaptation

The objective of the remaining component of the algorithm is to integrate the partitioning of the input space with the remainder of the learning algorithm. Specifically the partitions of the input space should reflect the frequency of the states visited by the controller. Furthermore, the frequency with which specific states are visited is likely to change as a function of the controller's progression through the various stages of training. This means that the clustering process is performed twice: once before pre-training and once after pre-training is completed.

The final amendment to the approach originally used [5] is to incorporate a fully stochastic weight change process (identified as the SWS-SWS algorithm later). Specifically, the Solis and Wets stochastic weight updating process is employed [12]. This provides much better independence with respect to the initial conditions. It also avoids the requirement for a hidden layer, therefore reducing training time and further improving invariance to initial conditions. The overall training algorithm for the linear network is summarized as follows:

1. Clustering of the input space using an arbitrary force generator;
2. Pre-training of the CRITIC network for a constrained set of initial conditions;
3. Pre-training of the ACTION network for a constrained set of initial conditions;
4. Re-clustering of the input space using the pre-trained ACTION network responses;
5. Repeat steps (2) and (3) for the complete set of training conditions.

3. Defining the Networks

In the following two formulations of the Mixtures of Experts (MoE) architecture are derived such that they fit within the context of SRV reinforcement learning. One uses the MoE methodology explicitly for the CRITIC and a Soft Weight Sharing approach for the ACTION, first introduced by Nowlan [11];

(SWS–MEnonlin). This produces a training algorithm based on a mixture of gradient and stochastic processes. A further generalization provides a purely stochastic training algorithm using the method of Solis and Wets [12]; for both ACTION and CRITIC networks (SWS–SWSnonlin). Two other modifications are investigated, in this case the experts and gates of the MoE architecture do not have a hidden layer, herein referred to as SWS–MElin and SWS–SWSlin respectively. The section is completed with a summary of the unsupervised learning algorithm used to perform the partitioning of the input space.

3.1. Non-linear Modular Training Algorithm

3.1.1. ACTION network

The aim of the ACTION network is to interact with the environment in such a way that the reinforcement signal is maximized. Actions of the former network are examined around a mean value $\hat{y}$ and a variance $\sigma(t)$ [1]. This variance is estimated by the equation:

$$\sigma(t) = \frac{k}{2}(1 - \tanh(p(t))$$

The scale parameter k is selected as a function of the dynamic range of the application domain; two in the case of the pole-balance problem, while $p(t)$ is the CRITIC network output. The stochastic response of the ACTION network is a function of:

$$\hat{y}(t) = N(y_i(t), \sigma(t))$$

where i indicates the expert and $N(a, b)$ is a normal random variable with mean a, variance b. To reiterate, the aim of the optimization network is to maximize reinforcement, $r(t)$, where the predictor responsible for deriving this, is based on a suitable regressor network. However, previous applications of reinforcement learning demonstrated the significance of partitioning the input space [1, 2]. As indicated above this is to be incorporated using a combination of the Mixtures of Expert (MoE) networks paradigm [4] and a very coarse partitioning of the input space (in the following application, five Gaussian basis functions per input). Control over the plant is expressed in terms of a normal random variable, with density function:

$$f(\hat{y}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\tfrac{1}{2}\left(\frac{\hat{y} - y_i}{\sigma}\right)^2\right\} \qquad (1)$$

Since the reinforcement learning algorithm does not provide any explicit information about a target response, weight update equations are a modified version of Gullipallis' weight change equation [3]. The modifications will focus on introducing stochastic updates on a Soft Weight Sharing basis [11] with a penalty term for large weights [12]. As a result the chain rule is formed as follows:

$$\frac{\partial r}{\partial y_i} = \frac{\partial r}{\partial c}\frac{\partial c}{\partial y_i} \tag{2}$$

The first term of equation (2) is the infinite horizon temporal difference expression [4]:

$$\frac{\partial r}{\partial c} = \big(r(t) + \gamma p(t) - p(t-1)\big) \tag{3}$$

where $r(t)$ is the reinforcement signal of the environment, and γ is the discount constant associated with the horizon (being 0.95). The last term in equation (2) is the partial derivative of the Soft Weight Sharing (*SWS*) cost function [11] with the penalty term for large weights defined by [13]. The main reason for employing *SWS* as a cost function for the ACTION network, as opposed to the form used in the original modular network [4], is to maintain structural simplicity. Hence with respect to (1) and (2):

$$c = \sum \frac{1}{2}(y_i - \hat{y})^2 - \ln \sum g_i \exp\left(-\frac{\left(\hat{y} - y_i\right)^2}{2\sigma^2}\right) + \lambda \sum \frac{\frac{y_i^2}{R^2}}{\left(1 + \frac{y_i^2}{R^2}\right)} \tag{4}$$

That is, the stochastic weight update change for the neural network output layer using reinforcement learning algorithm in the modular environment is:

$$\frac{\partial r}{\partial y_i} = \frac{\partial r}{\partial c}\left\{(y_i - \hat{y}) - \alpha h_i\left(\frac{y_i - \hat{y}}{\sigma^2}\right) + \lambda \frac{2\left(\frac{y_i}{R^2}\right)}{\left(1 + \frac{y_i^2}{R^2}\right)^2}\right\} \equiv \delta(t, t-1) \tag{5}$$

where λ is a suitable value constant, h_i is the *posterior* probability and R^2 is the maximum value f can take; in the case of the pole-balancing problem 10. A reasonable cost function is obtained by multiplying the second term of the equation by a constant; $\alpha = a\sigma$, where a is a small positive number. The effect that this term has in the cost function is significant because it keeps the values of equation 4 away from extremes. As a result training is more effective, while network performance improves significantly. Equation (5) is used to update the weights of the output layer of the ACTION network, while the weight changes in the hidden layer of the experts are calculated by:

$$\Delta w_{ij}^{(l-1)}(t) = \eta \delta(t,t-1) w_i f'\left(net_i^{(l-1)}\right) y_j^{(l-2)}(t-1) \tag{6}$$

The modular network will be compared against an MLP equivalent in order to determine the base line performance and the amount of improvement achieved by the proposed scenario. As far as the ACTION network is concerned the weight update equation for the non-modular network is given by [7]:

$$\frac{\partial r}{\partial y} \approx \left[r(t) - p(t) \right] \left[\frac{\hat{y}(t-1) - y(t-1)}{\sigma(t-1)} \right] \tag{7}$$

The gate network in a modular environment is responsible for allocating experts to specific regions of the input space. In effect this means that different experts are associated with different conditions in the state space. If no hidden layer exists, a gate network response using the winner-takes-all algorithm ensures that only the expert and gate weights corresponding to such a winning node are updated. These changes are described by the following equation [3, 4]:

$$\begin{aligned} \Delta w_i(t) &= \eta \frac{\partial l}{\partial w_i} = \eta \frac{\partial l}{\partial u_i} \frac{\partial u_i}{\partial (net)} \frac{\partial (net)}{\partial w_i} \\ &= \eta \left(h_i(t-1) - g_i(t-1) \right) f'(net(t-1)) x_i^{(l-1)}(t-1) \\ &= \eta \delta(t, t-1) x_i^{(l-1)}(t-1) \end{aligned} \tag{8}$$

3.1.2. MoE CRITIC network

The other network using a modular architecture resides in the CRITIC element of the SRV network. As indicated above, the objective of the CRITIC is to predict the significance of the current ACTION network response and discount this against the time horizon of the temporal difference method [4]. The error function for this network uses the same formulation as equation (3). Weight updating for the output layer of the experts residing in the CRITIC is calculated using a derived application of the modular network objective function [6]:

$$\begin{aligned} \Delta w_i(t) &= \eta \frac{\partial l}{\partial w_i} = \eta \frac{\partial l}{\partial y_i} \frac{\partial y_i}{\partial w_i} = \\ &= \eta h_i(t-1) e_i(t-1) x_i^{(i-1)}(t-1) \equiv \eta \delta(t,t-1) \end{aligned} \tag{9}$$

3.1.3. SWS CRITIC network

The above defined SWS–ME update relation is partially stochastic. SWS–SWS however, employs a *SWS* equivalent of the ACTION, for the CRITIC, which is defined from Nowlan's Soft Weight Sharing methodology [8]:

$$c = \sum \frac{1}{2} \hat{r}^2 - \sum \ln \sum g_i e^{-0.5\hat{r}^2} \tag{10}$$

The first term represents misfit, while the latter describes data complexity. The CRITIC error function $\hat{r}$ appears in both components of (10), while g_i is the gate network contribution. The latter also uses the log-likelihood function used by the gate network. For the hidden layer the process is similar to the one described by equation (6) for both versions, hence:

$$\Delta w_{ij}^{l-1}(t) = \eta \delta(t, t-1) w_i(t-1) f'\left(net_i^{(l-1)}\right) x_j^{(l-2)}(t-1) \tag{11}$$

To summarize the equations describing the SWS–MEnon-lin and SWS–SWSnon-lin algorithms are (1)-(9) and (11), and (1)-(8) and (10)-(11) respectively.

3.2. Linear Modular Training Algorithm

The above represents the learning algorithm for the combined SRV–MoE SWS architecture. In section 4 this is implemented with MLP experts and found to be sensitive to the initial conditions of the free parameters. In order to minimize this effect a fully stochastic learning algorithm (as opposed the mixed stochastic gradient descent regime described in section 3.1) is introduced, the MLPs are replaced with linear perceptrons, and data driven partitioning used instead of *a priori* partitioning.

Beginning with the latter requirement, many methods have been proposed from simple competitive learning, to EM type algorithms. Within this contest, most emphasis is attributed to the integration of an overall system; hence the clustering activity is to be achieved with a minimal computational cost. To this end competitive learning is employed but on two occasions. In the first case the method is used to identify initial locations for the clusters based on data derived from a sinusoidal forcing function. On the second occasion it is used to adjust the locations of the centers in accordance with forcing functions provided by the pre-trained ACTION network. Specifically, competitive learning using the insider role is employed (Hebbian weight adaptation with a forgetting factor) as

$$\Delta w_{kj}(t) = \alpha y_k(t)\left[cx_j(t) - w_{kj}(t)\right] \tag{12}$$

Estimation of the variance associated with the RBF centers is now determined directly from their location in accordance with:

$$\sigma = \frac{d}{\sqrt{2M}} \tag{13}$$

where d is the average distance between RBF centers and M is the number of centers. This clustering process is applied in steps one and four of the overall algorithm (section 2.2). Weight update for the ACTION network is derived solely using the *SWS* algorithm, while the CRITIC uses both *ME* (SWS–MElin) and *SWS* (SWS–SWSlin), defined by equations (1-9 excluding 6) and (1-8 excluding 6 and 10) respectively, while both including equations (12, 13) responsible for placing the RBF centers.

The second modification involves replacing the MLP experts with a single layer of non-linear perceptrons, trained using the stochastic algorithm of Solis and Wets [12]. This has the advantage of directly supporting the SRV stochastic algorithm of the reinforcement learning scheme, as well as dropping the requirement for adapting a hidden layer. By doing so the effect of initial conditions of the free parameters on network are minimized. Specifically the Solis and Wets weight updating algorithm consists of the following 3 steps [12]:

i) If $E(w^k + \Delta w^k) < E(w^k)$ then $w^{k+1} = w^k + \Delta w^k$.

ii) If $E(w^k - \Delta w^k) < E(w^k) \leq E(w^k + \Delta w^k)$ then $w^{k+1} = w^k - \Delta w^k$.

iii) For the case that none is satisfied $w^{k+1} = w^k$.

where E is the error as a function of weight, w^k, and the corresponding weight change Δw^k at the *kth* step.

4. Performance Evaluation

4.1 Pole-Balance problem

As indicated in section 1 evaluation of the proposed modular SRV ACTION–CRITIC network is judged against the pole-balancing reinforcement problem identified in [14] as a suitable benchmark problem for non-linear control. This benchmark has been widely used for assessing the capability of different reinforcement learning methodologies [1, 2, 15].

The inverted pendulum plant consists of a cart that is free to travel in either direction along a horizontal axis. Attached to the top surface of the cart is a pendulum. This has two 'rest positions' the first is horizontal with the plane of cart movement, and represents the failure state. The second is vertical to the plane of cart movement and is considered to be the ideal solution state. There are four input variables: linear position of the cart alone the horizontal axis, *x*, where this is measured in meters from the center of movement; $\dot{x}$ is the corresponding linear velocity measured in m/sec; angle of the pendulum, θ, measured in degrees from the upright position; and angular velocity of the latter $\dot{\theta}$ measured in rad/sec. The only parameter that the neural network can adjust to balance the pendulum (plant) is a force *f* (measured in *N*) and applied to the cart along the horizontal axis. The equations of motion governing the plant behavior are:

$$\theta(t+1) = \theta(t) + \Delta\dot{\theta}(t)$$

$$\dot{\theta}(t+1) = \dot{\theta}(t) + \Delta \frac{mg\sin\theta(t) - \cos\theta(t)\left[f(t) + m_p l\left(\dot{\theta}(t)\pi/180\right)^2 \sin\theta(t) - \mu_c \operatorname{sgn}\left(\dot{x}(t)\right)\right] - \frac{\mu_p m\dot{\theta}(t)}{m_p l}}{(4/3)ml - m_p l\cos^2\theta(t)}$$

$$x(t+1) = x(t) + \Delta\dot{x}(t)$$

$$\dot{x}(t+1) = \dot{x}(t) + \Delta \frac{f(t) + m_p l\left[\left(\dot{\theta}(t)\pi/180\right)^2 \sin\theta(t) - \dot{\theta}(t)\pi/180\cos\theta(t)\right] - \mu_c \operatorname{sgn}\left(\dot{x}(t)\right)}{m}$$

For these equations the values of constants are:

1) $g = -9.81 m/sec^2$ for the acceleration of gravity.

2) $m = 1.1\ kg$, for the mass of the pole and cart combined.

3) $m_p = 0.1\ kg$, for the mass of the pole.

4) $l = 0.5m$, being half pole length.

5) $\mu_c = 0.0005$ the coefficient of friction of the cart on the track.

6) $\mu_p = 0.000002$, the coefficient of friction of the pole on the cart.

7) $\Delta = 0.002$, Euler approximation sample step size.

The constraints that define the operating window (non-failure condition) of the plant are: $\pm$12° degrees for the angle; horizontal range of operation of $\pm$2.4 meters, while force must not exceed $\pm$10

N. If any of these parameters are exceeded, the reinforcement signal from the environment is -1 indicating failure, while for all the other conditions it is 1, indicating success.

4.2. Network Training and Test

A training cycle is conducted using the 6 input patterns of table 1. Training continues until *all* patterns have met the stopping criteria. Generalization is tested by assessing performance on 32 unseen initial conditions. These conditions are divided into two categories: those with the same horizontal displacement, and those with the same angle. In case of the former, position of the cart is considered to be +1m while the angle varies between $\pm$ 10° in increments of 2.5° degrees, repeating the test for a position of –1m. In the second category the angle remains constant at 5°, while position varies between $\pm$ 2.2 meters using increments of 0.6 meters, again repeating the test for an angle of –5°.

This approach differs from that of most previous work in that the initial condition was either always 0 for all parameters, in which case the network in effect learns-to-learn nothing [2, 7]. Alternatively, the initial input conditions are initialized to different values after each failure of the training algorithm, but training is considered complete when the first initial condition is learnt [1, 15].

The following study is in two parts. The first summarizes the performance achieved without a modular network in the respective ACTION and CRITIC elements and *a priori* partitioning of the input space. Instead an MLP is used in each. These results provide a performance base-line for assessing the significance of the modular schemes proposed in section 3. Part two assesses the performance of the hierarchical modular schemes: SWS–ME and SWS–SWS. Two scenario's are considered in each case, the first in which linear MoE are used and the second in which the MoEs are allowed to have a hidden layer.

In each case performance will be assessed in terms of: robustness to the initial conditions of the network; iterations to failure; and computational requirements during training. In terms of an ideal performance goal, both short training times and good generalization ability are sought.

4.3. MLP Performance

In this section the performance of SRV ACTION–CRITIC networks using MLP and linear networks in the ACTION–CRITIC elements is assessed. Weight updating in the MLP and linear networks is performed using equation (7) for the ACTION and standard BP in the case of the CRITIC network. Figure 2, columns

1 and 3, summarize performance of the MLP scenarios in terms of the number of iterations before failure on the training set (1st quartile, median and 3rd quartile). At best most non-linear MLP based networks avoid failure for a few thousand iterations. Failure occurs sooner in the case of the linear scenario. Table 2, row 1 column 3, provides a T-test statistic of the similarity between the two methods; the results are statistically independent with the non-linear case typically performing better than the linear case. In terms of training requirements on average it takes about 400 iterations for the MLP to converge while the linear network requires about 200 iterations; figure 3, columns 1 and 4. However, computationally, the linear case is much faster; Figure 4, columns 1 and 4.

Generalization is assessed by examining network performance over the above 32 unseen input conditions. Figure 2, columns 3 and 4, and Table 2, row 1 column 4, indicates that the both the linear and the non-linear MLP networks balance the pendulum for a period consistent with that achieved during training. Given the known non-linear characteristics of this problem, this is expected. Test results for these architectures do not show a significant degree of performance discrepancy between the different initialization conditions. However, none of the training or test conditions observed represent a satisfactory control action. That is to say, the action for the controller in both cases is unidirectional, thus leading to the failure criteria. The networks investigated in this section therefore do not provide a sustainable control action on any initial pendulum/ cart position.

4.4. Modular Network Performance

The previous section indicated that the problem was not linearly separable using the combination of short training cycles and SRV ACTION–CRITIC paradigm alone. Furthermore, the MLP version of this architecture (i.e. incorporating non-linear learning capabilities) was not able to sufficiently generalize the problem from the training set and convergence criteria employed. This is not to say that an MLP based SRV ACTION–CRITIC network is completely unable to solve the problem as posed, but that the resources necessary to achieve this are in excess of those available in practice. By employing the modular scheme under the same training conditions, an assessment of the significance of the proposed hierarchical modular learning algorithm is made.

The combination of the MoE architecture and partitioning of the input space leads to a larger number of design decisions necessary to specify the network architecture. In this case the number of MoE experts and input space partitions is constrained to be the same. Empirical evaluation of the network for different numbers of experts/ partitions quickly identified this design parameter through evaluation of the MoE gate response. Too few partitions lead to an over reliance on the properties of

the experts. Too many partitions lead to a high degree of overlap in the gating network, thus causing poor partitioning between experts.

In this evaluation, two different versions of the algorithm for training the modular networks are used. One employs full *SWS* on both components of the SRV ACTION–CRITIC networks (SWS–SWSlin/ non-lin), while the other uses a combined *SWS–ME* scenario (SWS–MElin/ non-lin) [16]. As in the previous section evaluation begins with network performance over the training patterns.

4.4.1. Testing Network Learning Abilities.

Evaluation of network performance over the *training* patterns follows a different philosophy from the one mentioned in [15]. There, a network is effectively considered trained if a balance state is visited from *any* single start state within a total test period of 5×10^4 iterations. Under this convergence criteria all SWS–ME/ SWS networks satisfy such a generalization test.

In the case of all the networks employing the proposed modular scheme, some cart/ pole conditions are observed to be balanced for over 3×10^4 iterations, while others manage only a few thousand. Examination of the extended cases reveals that control parameters vary in a sinusoidal manner, where this is replicated in all the measured parameters. Specifically, figures 5, 6 (network 1, x = 1m, $\theta = -5°$), show that the angle of the pendulum and corresponding velocity vary at a constant frequency and amplitude. On the other hand, respective cart position and velocity operate as decaying sinusoids, figures 7, 8. The controller force which gives rise to this condition, however, does not simplify to such a simple analysis, figure 9. This operation of the input parameters is taken to imply that for the particular initial input condition, the network has captured plant dynamics accurately enough to provide an extended control action, hence the pattern is considered 'learnt'. Similar findings are observed for the all the input conditions that reach the 3×10^4 limit. Analysis of the instances which fail before reaching this condition reveals that not *all* the control parameters vary in a sinusoidal manner, thus encountering one of the failure conditions, section 4.1. Specifically, the linear acceleration of the cart is observed to always have a unidirectional component, making it inevitable that the plant will reach the positional failure condition. It is also apparent from table 3 that the majority of the networks tested do not identify control actions reaching the threshold, implying dependence on initial positional conditions. Since analysis of control actions reaching the 3×10^4 iteration limit provide stable control, this will be used as a threshold distinguishing the success or failure of the network.

Performance of the various *SWS–ME* and *SWS–SWS* algorithms is summarized in figure 2 (columns 5 to 12), table 2 (confidence intervals) and 3 (instances reaching 3×10^4 iterations). It is immediately apparent that there is no longer a preference for the inclusion of a hidden layer. In the case of the SWS–ME pair, there is no significant difference in the performance between the two methods (linear or non-linear). The SWS–SWS pair however has a significant preference for the linear network scenario. Contrasting the SWS–ME and SWS–SWS algorithms indicates that the non-linear cases are indistinguishable, but the linear variant of the SWS–SWS algorithm (SWS–SWSlin) is significantly better than any of the other networks tested.

In terms of necessary training requirements, figures 3 and 4, it is observed that all modular networks consume more training resource than the non-modular cases. The main reason for this however, is that included within this estimation are the iterations necessary to train the unsupervised network performing the partitioning of the input space. This activity is not performed at all in the case of the above MLP cases. As observed previously, the linear cases (SWS–MElin/ SWSlin) have a higher iteration count, but lower actual training time. It is interesting to note however, that the SWS–ME algorithms take a significantly longer time to converge than the corresponding SWS–SWS cases; where this is most pronounced in the number of iterations i.e. typically double the SWS–SWS requirement.

4.4.2. Testing Networks Generalization Abilities.

The generalization test focuses on the ability of the network to identify the previously described sustaining control action on a *set of unseen* cart/ pole conditions. This is different from the criteria used in [15] in which the generalization test is limited to the network lasting 1000 iterations without encountering a failure condition on any one unseen condition. Using such a test for generalization would result in *all* networks fulfilling the criteria.

In all cases both the performance under test and training set patterns remains consistent, hence the above observations also hold for the case of unseen initial conditions; figure 2. Importance of initial conditions of the free network parameters to final network performance is evident, since some networks manage to repeatably reach the threshold while others fail; table 3.

In summary, the combination of a completely stochastic training algorithm with linear experts in the SWS–SWS algorithm out performs the other methods as measured by the 99% confidence interval; table 2. Furthermore, the SWS–SWS algorithms typically requires half as many iterations to

reach convergence than the corresponding SWS–ME cases, whilst at worst matching the performance of the SWS–ME case. The MLP based methods are unable to generalize at all, making their faster training time inconsequential.

4.4.3. Gate Assessment

The ability of the devised algorithms to associate individual experts with specific regions is examined by observing the response of the MoE gate network. For the non-linear architecture, gate response of both *SWS–ME* and full *SWS* models, are shown in figures 10 and 11. It is apparent that no scenario manages to explicitly partition, that is focus one expert on a unique region for either the ACTION or the CRITIC network. However, it is noted that expert 3 always contributes at least 40% to the final action while the others assist, except for the center region where it solely controls the plant. This implies that for this architecture successful control actions are achieved by linearly combining non-linear networks.

In the case of the linear network (SWS–MElin/ SWSlin) gate response, figures 12 and 13, three partitions occur. Typically two networks contribute to each of the extremes (expert pairs (1,4) and (2,5)), whereas one network dominates in the central location (expert 3). That is a non-linear mixture of linear networks, a predictable result given the inherent non-linearity of the plant.

5. Related Work

In terms of the architecture employed the most significant related network is that of the hierarchical CMAC network developed by Tham [17]. Here a MoE network is used as the action generator with CMAC networks as the experts and gate. The method used to partition the input spaces is therefore very different. CMAC emphasizes efficient covering of the entire input space. The RBF based clustering used here represents a wholly data driven partitioning of the input space, such that only the areas of interest are included (vis-à-vis the training set). Moreover, Q learning as oppose to an ACTION–CRITIC architecture is used. Furthermore, the EM algorithm is used to adapt the network parameters [18], rather than the completely stochastic approach of the preferred SWS–SWS method proposed here. Finally, in the application discussed by Tham, intermediate rewards are explicitly incorporated, thus performing associations over long temporal horizons is not actually necessary.

Millán carries the use of enriched reinforcement information much further in his work with a modular architecture [19, 20]. Here sensors are applied to the application (autonomous robot and a

robot arm respectively) in order to create a grid capable of feeding back information about the state of the environment. Several metrics are then built into the reinforcement signal, with updating performed on a step-by-step basis. In terms of the architecture, Gullapalli's interpretation of the ACTION–CRITIC architecture is again used as the starting point. However, a tessellation based constructive algorithm is used to create the partitions [19]. Furthermore, the grid of the sensors is then used interactively to ensure efficient exploration of the input space. In the case of [20] more traditional adaptive process is employed based on the method of truncated temporal differences and Williams's REINFORCE algorithm.

Lin and Lee take the more general motivation (as is the case with the work conducted here), applying a neuro-fuzzy system to Gullapalli's interpretation of the ACTION–CRITIC architecture [7]. The resulting network works in a continuous environment (as opposed to the quantized actions of the above) and employs Sutton's temporal difference learning with gradient decent to adapt the free parameters. The network is initialized with a random set of IF–THEN rules, which when applied to the pole-balancing problem yields 35 rules or 31 clusters (as opposed to the 20 partitions used by the SWS-ME/ SWS methodologies).

6. Conclusion

This paper details the extension of Gulapalli's SRV reinforcement learning to include modular networks in the ACTION and CRITIC partitions. By combining this with unsupervised clustering of the input space, for the positioning of the RBF centers used in the gates and experts, a three layer partitioning of the architecture is provided. This facilitates the decomposition of the learning procedure into the individual adaptation of sub-components of the overall network. The resulting architecture is demonstrated to have the ability to identify partitions within the context of a temporally discounted non-linear control problem, without *a priori* partitioning of the input space, additions to the reinforcement signal or resulting in tens of partitions. A comparison is made between the proposed MoE scenario and the equivalent architecture with MLP ACTION and CRITIC. The MLP based system is unable to 'learn' any pattern within the short training cycles employed. The modular linear experts architecture however, is shown to have the capability to master the non-linear dynamics of the system in several cases. Similar are the findings for the non-linear equivalent, which performs control actions in a less effective manner. Furthermore the linear SWS–SWS topology out performs all of the

networks assessed at the 99% confidence interval, whilst requiring half the training time of the hybrid gradient–stochastic learning algorithm.

It is also emphasized that the test conditions are more difficult than usually reported due to the non-zero parameter initialization and more stringent condition for generalization (without which 100% generalization is achieved). It is acknowledged that both architectures still lack the ability to efficiently store temporal information. That is to say, the system still relies on matching combinations of the input space partitions and associating these states with temporal difference weighted outcomes. This problem will be addressed in future research.

References

1. Anderson C.W., "Strategy Learning with multi-layer connectionist representations," Proceeding of 4th International Workshop on Machine Learning, pp 103-114, 1987.
2. Barto, R.S. Sutton, C.W. Anderson, "Neuronelike Adaptive Elements that can solve Difficult Learning Control Problems," *IEEE Trans. SMC part B*, vol. 13, no. 5, pp 834-847, 1983.
3. Gulapalli V., "A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions," *Neural Networks*, 3, pp 671-692, 1990.
4. Sutton S., "Learning to Predict by the Methods of Temporal Difference," *Machine Learning*, vol. 3, pp 9-44, 1988.
5. Heywood M.I., Chan M-.C., Chatwin C.R., "Application of stochastic real-valued reinforcement neural networks to batch production rescheduling," Proc. Inst. Mech. Engrs., Vol 211 Part B, pp 591-603, 1997.
6. Jacobs, M. I. Jordan, "Learning Piecewise Control Strategies in a Modular Neural Network Architecture," *IEEE Trans. on SMC part B*, vol. 23, no. 2, pp 337-345, March/April 1993.
7. Lin C.-T, Lee C.S.G., "Reinforcement Structure/ Parameter Learning for Neural Network Based Fuzzy Logic Control System", *IEEE Trans. on Fuzzy Logic*, IEEE, USA, February 1994, vol. 2, no. 1, pp 46-63, 1994.
8. Lin L-.J., "Self Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching," Machine Learning, 8, pp 293-321, 1992.
9. Werbos J. Paul, Approximate dynamic programming for real-time control and neural network. In Handbook of Intelligent Control, White A. David, Sofge A. Donald (ed), Van Nostrand-Reinhold, New York, pp 493-524, 1992.
10. Kehagias A., Petridis V., "Predictive Modular Neural Networks for Time Series Classification," Neural Networks, 10(1), pp31-50, 1997.
11. Nowlan J., Hilton G. E., "Simplifying Neural Networks by Soft Weight-Sharing" *Neural Computation*, vol. 4, pp 473-493, December 1992.
12. Solis F.J., Wets J.B., "Minimization by random search techniques," Mathematics of Operations Research, 6, pp 19-30, 1981.
13. Weigend , B. A. Huberman, D. E. Rumelhart, "Predicting the future : A connectionist approach", *International Journal of Neural Systems*, vol. 1, no. 3, pp 193-209, 1990.

14. Anderson C.W., Miller W.T., "Challenging Control Problems," *Neural Networks for Control*, Appendix A, (ed), Miller W.T., Werbos P.J., MIT Press, 1990.
15. Moriarty E. David, Miikkulainen Risto. Efficient Reinforcement Learning through symbiotic Evolution. Machine Learning 1996; 22: pp11-32.
16. Paraskevopoulos V, Heywood M, Chatwin R. Modular SRV Reinforcement Learning: An architecture for non-linear control. IJCNN'98; Anchorage, Alaska, May 1998; pp 2034-2038.
17. Tham C.K. Reinforcement learning of Multiple Tasks using a hierarchical CMAC architecture, Robotics and Autonomous Systems, 1995, 15 (4) pp 247-274.
18. Jordan M.J., Jacobs R.A., Hierarchical Mixtures of Experts and the EM algorithm. Neural Computation, 1994, 6, pp 181-214.
19. Martín P., Reinforcement learning of goal-directed obstical-avoiding reaction strategies in an autonomous mobile robot, 1995, 15(4), pp 275-300.
20. Martín P., Millán J.R., Learning reaching strategies through reinforcement for a sensor-based manipulator, Neural Networks, 1998, 11 (2), pp 359-376.

Table 1: Neural network training patterns.

Angle in degrees	Position in meters	Angular velocity rad/sec	Linear velocity m/sec
-2.2	11	-18	9
-1.6	8	-13.0909	6.5455
-1	5	-8.1818	4.0909
1	-5	8.1818	-4.0909
1.6	-8	13.0909	-6.5455
2.2	-11	18	-9

Table 2: Control activity — Confidence interval tests

Networks compared	T-test	Data Set	
		Train (%)	Test (%)
MLP	Linear MLP–v–Non-linear MLP	0.014	0
Mixed	Linear MLP–v–Version 1	0	0
	Non-linear MLP–v–Version 2	0	0
Modular	Version 1–v–Version 2	74	48
	Version 3–v–Version 4	4.1	0
	Version 1–v–Version 3	13	0
	Version 2–v–Version 4	88	93

Table 3: Percentage of cases reaching 30,000 Threshold

Algorithm	Training	Test
Linear SWS–ME	13%	16%
Non-Linear SWS–ME	8%	3%
Linear SWS–SWS	27%	31%
Non-Linear SWS–SWS	7%	8%

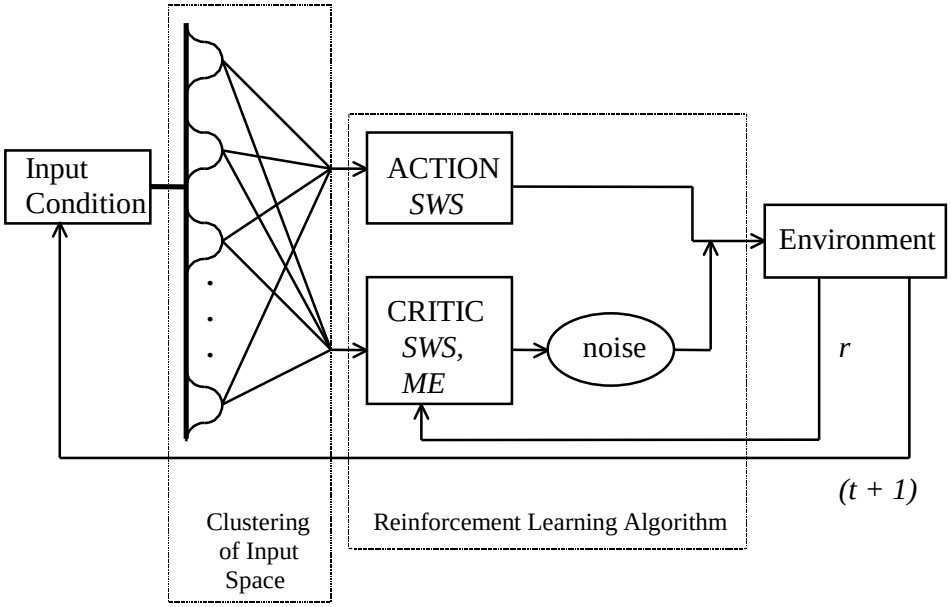


Fig 1: SRV-Modular network architecture.
CRITIC and ACTION elements composed of MoE architectures, SWS/ ME denotes the learning rule.

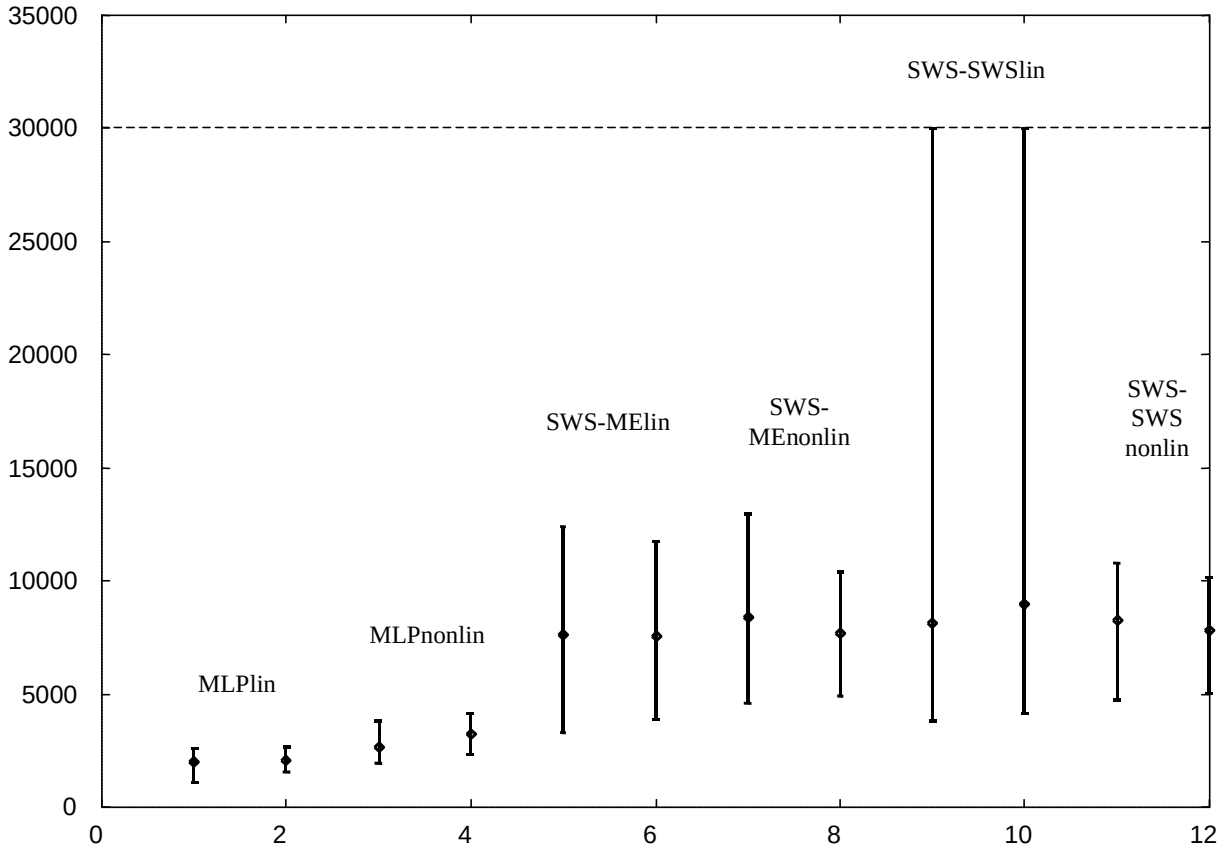


Fig 2: Network iterations before failure – 30,000 Threshold (1st Quartile, Median, 3rd Quartile).
Key: #1 linear MLP (train); #2 linear MLP (test); #3 non-linear MLP (train); #4 non-linear MLP (test); #5 linear SWS–ME (train); #6 linear SWS–ME (test); #7 non-linear SWS–ME (train); #8 non-linear SWS–ME (test); #9 linear SWS–SWS (train); #10 linear SWS–SWS (test); #11 non-linear SWS–SWS (train); non-linear SWS–SWS (test).

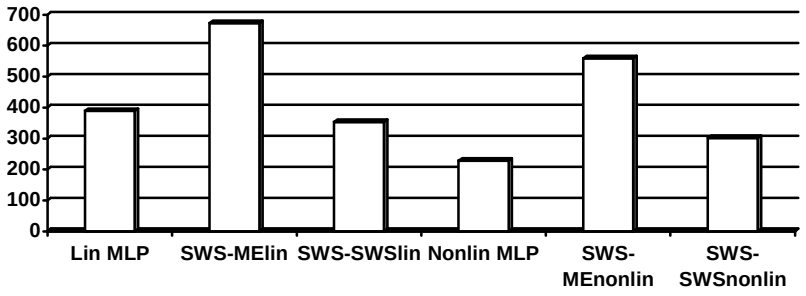


Fig 3: Mean iterations during training.

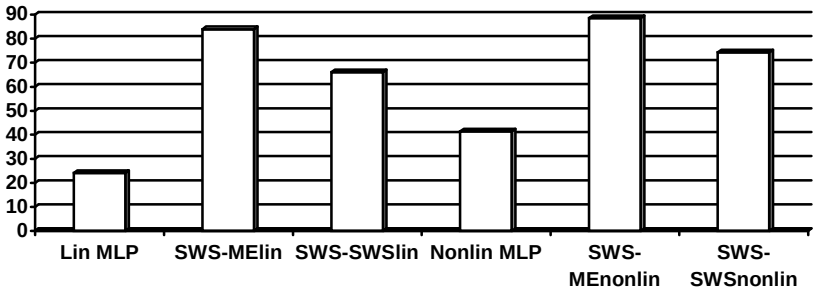


Fig 4: Mean simulation time (sec) during training.

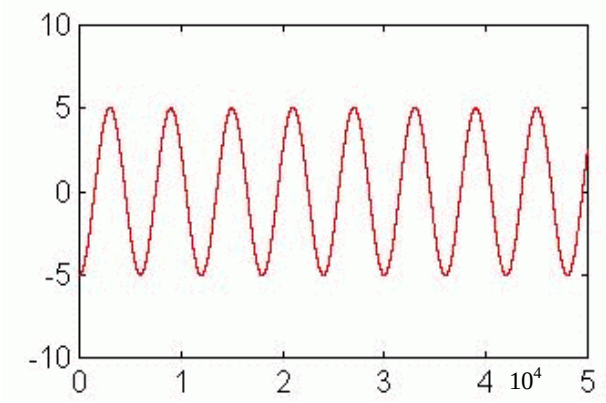


Fig 5 : Angle of the pendulum in degrees.

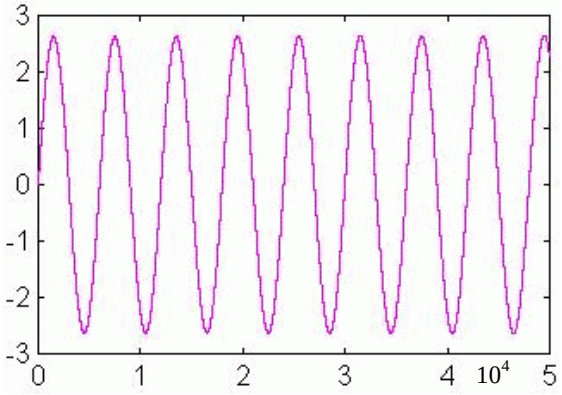


Fig 6: Angular velocity of the pendulum in rad / sec.

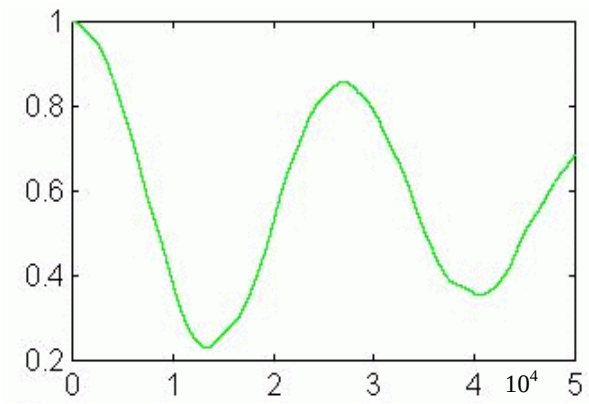


Fig 7: Position of the pendulum in meters.

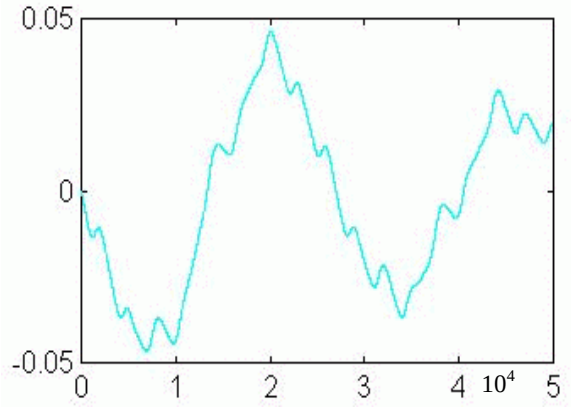


Fig 8: Linear velocity of the pendulum in meters / sec.

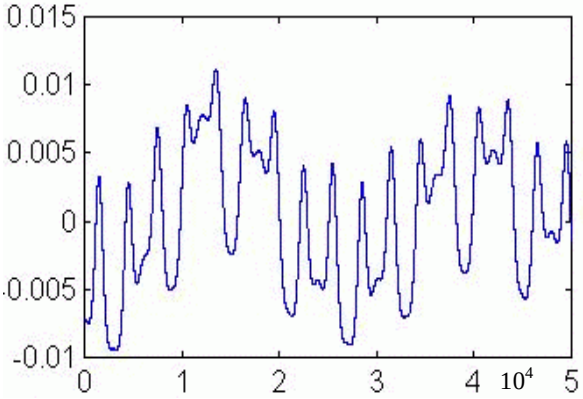


Fig 9: Force on the system in N.

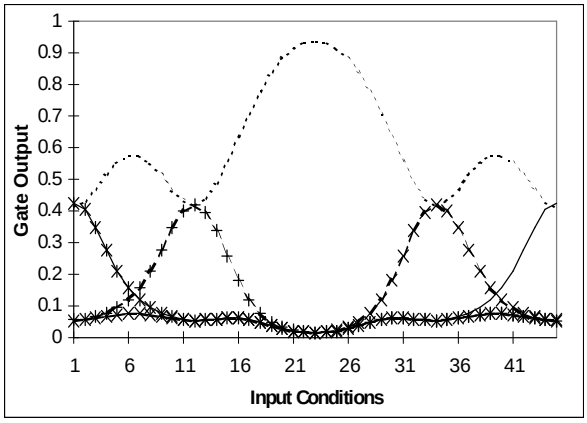


Fig 10(a): ACTION network.

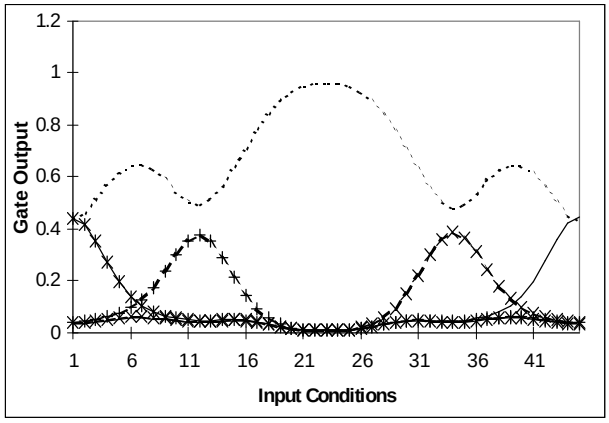
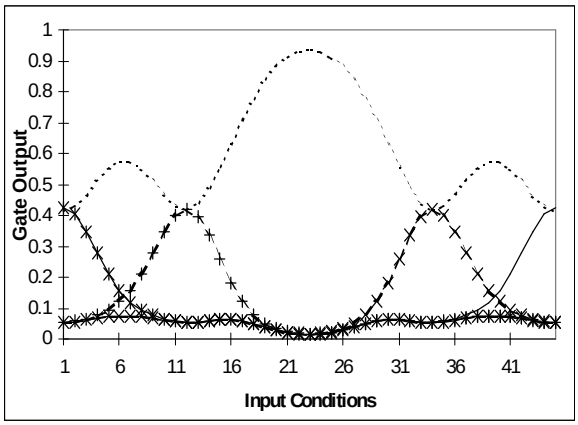


Fig 10(b): CRITIC network.

Fig 10: SWS-MEnonlin Gate Network Response.

Key: expert 1; expert 2; expert 3; expert 4; expert 5.



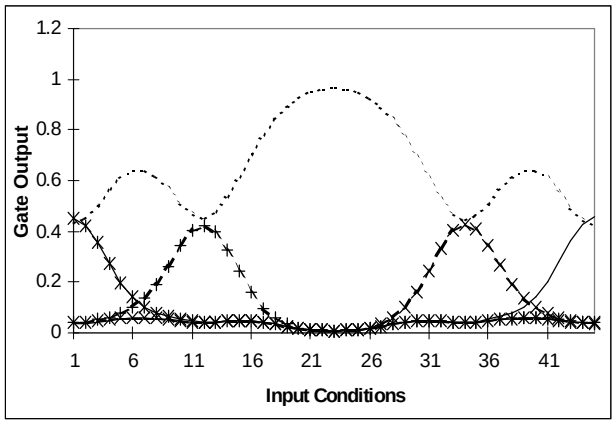
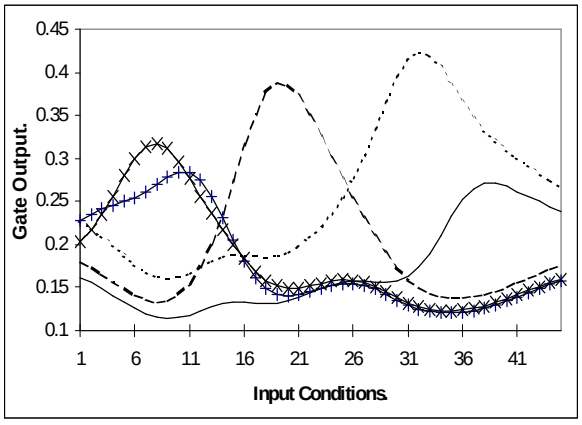


Fig 11(a): ACTION network. Fig 11(b): CRITIC network.
Fig 11: SWS-SWSnonlin Gate Network Response (key: as figure 10).



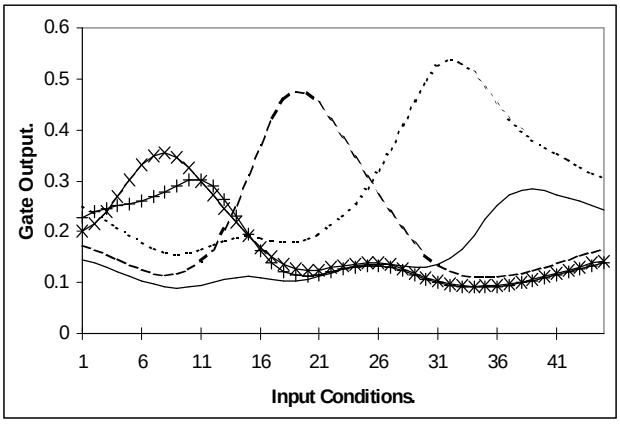
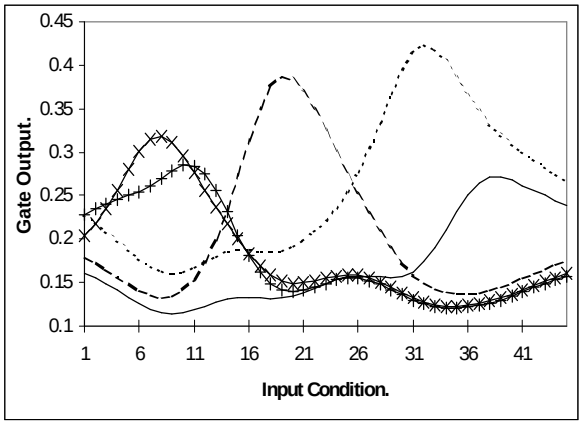


Fig 12(a): ACTION network. Fig 12(b): CRITIC network.
Fig 12: SWS-MElin Gate Network Response.
Key: expert 1; expert 2; expert 3; expert 4; expert 5.



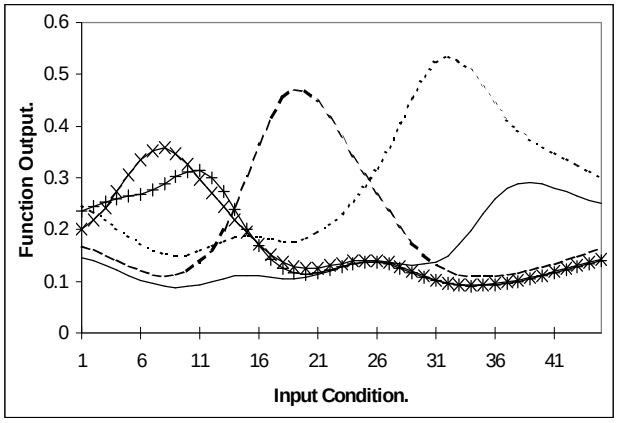


Fig 13(a): ACTION network. Fig 13(b): CRITIC network.
Fig 13: SWS-SWSlin Gate Network Response (key: as figure 12).