Direct Optimal Control using TD(λ) Mixtures of Experts

V. Paraskevopoulos⋆, M. I. Heywood⋆⋆, C. R. Chatwin⋆

⋆University of Sussex, School of Engineering, Falmer, Brighton, BN23 8QT. UK
⋆⋆Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax, Nova Scotia. Canada B3H 1W5

Abstract- *Real-time control of continuous valued plants using TD(λ) reinforcement learning is detailed. This problem is significantly more difficult then the case of a discrete control space as in bang-bang or Q-learning. The methodology employs a combination of Stochastic Real-Valued units, Mixtures of Experts and RBF partitioning To do so the significance of both Maximum-Likelihood and Square Error Cost functions are emphasised, as is provision for RBF co-variances during training. The resulting architecture is demonstrated on benchmark problems.*

1. Introduction

Optimal control requires the definition of a control policy from the behaviour of a plant, which is not necessarily defined in terms of a desired reference trajectory. Architectures applicable to optimal control problems are generally expressed in terms of direct and indirect schemes [1]. With direct approaches, the parameters of the controller are adapted to minimise some norm of the plant error. Indirect methods first model the plant and then derive the relevant control law from this model. Many examples are available for each of these methodologies, in recent years however, the model based/ reference approach to indirect control has received a lot of interest.

One of the reasons for an emphasis on indirect control is that a large body of previous work from classical control theory is available. Moreover, the direct approach to optimal control implies that there is no direct error function, hence a significantly more difficult learning problem. Such a perspective means that most emphasis of neural network methods has been towards indirect control and the satisfaction of stability criteria for regulation and tracking problems in which the trajectory is readily known *a priori*. However, the control problem as a whole may be taken a stage further by including determination of what is typically considered *a priori* information. The problem has now became one of identifying a minimising function in a space of state and control trajectories, both of which are a function of time. Again several classical approaches are available e.g., Calculus of Variations and Dynamic Programming. The former is only applicable when a complete mathematical model of the plant is available. The latter also requires an analytical model, but provides a much more robust model of the system. Moreover the method has received considerable research interest from both the neural network and Dynamic Programming communities, the result of which has been techniques such as Approximate Dynamic Programming [2].

The defining property of such a system is the ability to adapt through *interaction* with the environment. This conforms to the direct method of optimal adaptive control where the feedback from the environment, *r*, is not in the form of an explicit error function, but a scalar, hence no sign or directional information is available. Such a function may take one of two forms: magnitude alone, or binary fail/ no fail information. Any learning system therefore has to first identify a utility function capable of providing a more descriptive cost function, $\hat{r}$, as well as optimising the present state. This gives rise to a class of methods called temporal difference learning, of which there are two general forms, TD(λ) [3] and Q-learning [4]. The latter method integrates both the estimation of the utility function and specification of the current action, but is specific to binary control functions, whereas the identification of control polices is the objective of this work. Moreover, TD(λ) enables the division of duties – estimation the utility function and selecting the current control policy – between two different networks.

The specific purpose of this work is to assess the applicability of highly partitioned feed-forward architectures to the identification of real-time (continuous valued) direct control context, where our initial motivations are provided by the recent interest in switching controllers [5]. To do so the Stochastic Real Valued (SRV) unit [6] is used to both provide a continuous valued output and support the ACTION–CRITIC partition of duties between selecting the current control policy and estimating the utility function (as in the AHC of Barto et al. [7]). ACTION and CRITIC are expressed as hierarchical Mixture of Expert (MoE) models, section 3.1, with Radial Basis Function (RBF) partitioning of the input state space, section 3.3. The adaptation of such an architecture typically takes the form of a probabilistic Maximum Likelihood model with Expectation Maximisation (EM) update procedure. Here however, an alternative cost function is employed, which when used with an annealed weight updating procedure provides a

trade-off between square error and Maximum Likelihood cost functions, section 3.2.

The paper begins with a summary of the TD(λ) temporal difference learning method, section 2. The proposed architecture is summarised in section 3, and performance assessed in section 4. Section 5 concludes the findings.

2. TD(λ) temporal difference learning

In the case of TD(λ), adaptive policy iteration takes place by dividing the task into two components, a predictor (CRITIC) and a regressor (ACTION), where this may have an explicit architectural embodiment; figure 1. The CRITIC produces a utility function, relating states of the environment to the predicted discount value (utility). The objective of the regressor is to maximise the utility function, as supplied by the predictor. The predictor learns to calculate specific policies through a tuple defining a transition of the environment between previous state, $s(t-1)$, current scalar reward, r, and the current state of the environment, $s(t)$ [2, 3, 4, 6, 7]. The adaptive rule takes one of four forms, depending on how far in the future a prediction of discounted pay-off is required [3], or as a general form

$$\frac{\partial C}{\partial \hat{r}} = \pi\, e_s$$

where π is the current policy as estimated for the temporal horizon anticipated and e_π is the eligibility trace (weighting over the temporal horizon in accordance with the received reinforcement).

The TD(λ) method however, suffers from poor scaling as the temporal period over which prediction increases, in much the same way that backpropagation through time also suffers an exponential scaling of the error term. Moreover, when ACTION and CRITIC are trained in tandem, significant difficulty is experienced in selecting appropriate learning parameters. Independent predictor and regressor adaptation is therefore appropriate in most cases, although theoretically sub-optimal. Moreover, approximations to TD(λ) have been proposed which significantly speed the evaluation of TD(λ) cases with non-zero regency factor [8].

In this work we are explicitly interested in the use of partitioned network architectures to facilitate learning of temporal difference policies.

3. TD(λ) Mixtures of Experts Model

As indicated above, we desire a direct controller applicable to continuous valued tasks in which partitioning plays a significant role in the architecture of the controller. Several examples of such a control system exist using general feed-forward architectures [9], fuzzy systems contexts [10], combinations of the latter two [11], or genetic algorithms [12]. The approach proposed here, however, directly incorporates the Mixtures of Experts (MoE) paradigm. By doing so, an explicitly probabilistic framework for training the network is availed, where this has been shown to provide several features of significance to time series modelling in addition to the above justifications [13]. Moreover, the use of *a priori* knowledge of any form is explicitly avoided. One of the motivations for this work was to assess the unaided ability of the neural method to solve problems in a real-time control context. The resulting architecture is most similar to the CQ-Learning methodology of Singh [14, 15], but applicable to the case of continuous valued outputs and makes use of a completely stochastic learning procedure. In the following, the Mixtures of Experts methodology is reviewed; sub-section 3.1. The resulting TD(λ) algorithm is described in sub-section 3.2.

3.1. Mixtures of Experts Architecture

The following is specific to the standard MoE architecture, where this is easily generalized to the case of multiple hierarchies [16, 15]. The objective of the network is to partition the input region between expert networks, such that individual experts (or sets of experts) become responsible for different polices of the control behaviour. A gating network defines the relation between experts. Specifically, each expert is a standard feedforward neural network (e.g. CMAC, RBF, MLP, linear network) producing a mapping $y_i = f_i(\boldsymbol{x})$, where i is the expert index. The gating network has as many outputs as there are expert networks, thus g_i is the gate for expert i. The purpose of the gate is to estimate the probability that input $\boldsymbol{x}$ was generated by expert network i. To provide such an interpretation the outputs from the gate are normalised to be positive and sum to unity, or

$$g_i = \frac{\exp(s_i)}{\sum_{j=1}^{K} \exp(s_j)} \qquad (1)$$

where s_i is the weighted sum of the inputs to output unit i, thus in the case of a linear network, $s_i = \sum_{j=1}^{M} w_{ij} x_j + b_i$. Hence (1) defines K competing probabilities as a function of the input, $\boldsymbol{x}$, where this competition is "soft". The overall output of the network is defined as the linear combination of the expected values from each expert, $\boldsymbol{y} = \sum_{i=1}^{K} g_i y_i$. Now if each expert is responsible for different pattern sets, then the g_i terms become binary during adaptation.

A cost function is now required, where the regimes for partitioning the experts is unknown

initially, hence an unsupervised learning context. A statistical framework of the maximum likelihood method is applicable, however, in which case it is necessary to assume a specific distribution for the measurement errors [16]. This means that a training pattern, d, is assumed to be created by a probabilistic process in which, for each pattern presentation, an expert is selected with *prior* probability g_i, i.e. based on input $\boldsymbol{x}$ without knowledge of the target. Thus, given a regression basis to function prediction/ approximation, then processes follow a statistical model of the form, $d = y_i + \varepsilon$, where y_i is a nonlinear function of the input and ε is a random variable. By assuming that ε has a Gaussian distribution, then the residuals of $d - y_i$ are also Gaussian, and (2) denotes the log-likelihood of generating a particular target vector d,

$$C = -\ln\left[\sum_{i=1}^{K} g_i P(d \mid x,\theta)\right] \qquad (2)$$

where θ are the free parameters of the model e.g. expert specific variance σ_i and weights $\boldsymbol{w}_i$; and $P(d \mid \boldsymbol{x}, \theta)$ is the expert specific conditional probability or

$$P(d \mid x,\theta) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(d-y_i)^2}{2\sigma_i^2}\right)$$

In this case each expert describes the mean for its specific density function, where this is a function of the input. The second parameter characterises the variance of the expert's Gaussian density function; where this is independent of the input, but specific to the expert[1].

The maximum-likelihood formulation also provides for explicit representation of the *posterior* probability through Bayes' rule, once both input and output are available, or

$$h_i = \frac{g_i P(d \mid x,\theta_i)}{\sum_{j=1}^{K} g_j P(d \mid x,\theta_j)}$$

Hence, in addition to the explicit partitioning of the learning task between multiple experts, the combined support for both *prior* and *posterior* probabilities gives the MoE methodology a significant advantage over global methodologies such as the multilayer perceptron (MLP). In particular the problem of adaptation is divided between a supervised element, which learns to predict the next value, d, and an unsupervised element, which identifies hidden regimes [13].

At this point the model has been characterised but the adaptive processes has not. Three basic options exist: gradient decent, as popularised by the back-propagation algorithm of MLP; the Expectation Maximisation (EM) algorithm, a general unsupervised algorithm for parameter adaptation [13-16], and; Deterministic annealing, an entropy minimisation procedure [17]. Direct application of gradient decent provides the following basic set of relations,

(1) updating expert weights: $\dfrac{\partial C}{\partial y_i} = \dfrac{h_i}{\sigma_i^2}(d - y_i)$

The typical difference relation (as seen in backpropation for example) is augmented by the *posterior* probability, h_i, hence modulating the significance of a weight change in proportion to the significance of expert i. Secondly, the MoE framework directly incorporates the significance of the error distribution associated with expert i. That is to say, when the predicted error σ_i is high then the significance of the error term, $d - y_i$, is discounted; whereas small predicted errors in the expert result in an increased sensitivity to the difference term.

(2) updating gate weights: $\dfrac{\partial C}{\partial s_i} = \eta(h_i - g_i)$

This implies that the parameters of the gating network are manipulated such that the *posterior* probability, h_i, is modelled using knowledge of the input, $\boldsymbol{x}$, alone.

(3) updating the variance of individual experts:

$$\frac{\partial C}{\partial \sigma_i^2} = \frac{h_i}{2\sigma_i^4}\left[(d - y_i)^2 - \sigma_i^2\right]$$

Where this effectively adjusts the variance of an expert, σ^2, to model the sample variance, $(d - y_i)^2$.

3.2. SRV Mixtures of Experts Architecture and TD(λ) learning

As indicated above, the TD(λ) paradigm represents a scheme for calculating delayed payoffs in a continuous valued environment and may be expressed in terms of a partitioning of the problem into separate predictor (CRITIC) and regressor (ACTION) networks [2, 6, 7]; figure 1. However, in order to provide a continuous valued output the ACTION–CRITIC components of the TD(λ) framework are formulated as the mean value of a recommended action, y, and a variance term, σ,

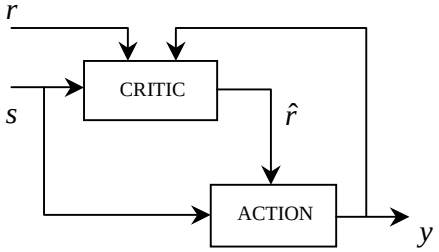

Fig 1: Action – Critic architecture used to partition the TD(λ) algorithm.

[1] Weigend shows that the independent estimation of variances for each expert has a significant effect on the overall performance of MoE models as applied to time-series predictive problems [13].

respectively; i.e. a Stochastic Real Valued (SRV) unit [6]. Hence, the overall response is described in terms of a Gaussian random variable,

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

with mean *y* and variance,

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

The scale parameter *k* is selected as a function of the dynamic range of the application domain; *i* indicates the 'winning' expert. The aim of the ACTION network is to maximise utility, $\hat{r}$, at the current time step using the control action, *y*. The CRITIC, however, is responsible for providing a better utility function given the current reinforcement signal, *r*(*t*). This is supported using a combination of the Mixtures of Expert (MoE) networks paradigm and a very coarse partitioning of the input space. There are therefore three levels of partitioning in the architecture (figure 2): separate CRITIC and ACTION networks; Mixtures of Experts paradigm (where this itself may consist of multiple layered partitions) and; Radial Basis Function (RBF) partitioning of the input state space.

The action network expresses actions in terms of a normal random variable, with density function,

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

The learning rule is derived from the stochastic gradient relation,

$$\frac{\partial r}{\partial y_i} = \frac{\partial r}{\partial C}\frac{\partial C}{\partial y_i} \qquad (3)$$

The first term on the right is the appropriate TD(λ) policy function [3] whereas the second term is dependent on the cost function of the regressor network, in this case the MoE architecture. As indicated in section 2 the cost function of a MoE model typically takes the form of a Maximum Likelihood estimator; c.f. equation (2). The basis for this decision is that, on a cross section of problems the algorithm functions well and enjoys an efficient update rule in the form of the EM algorithm [13-16]. However, an explicit regression context implies that retaining a cost function based on the square error is also justified. Here a different starting point is employed, in particular Nowlan's Soft-Weight-Sharing (SWS) scheme [18]. The initial objective is to incorporate penalties into the cost function regarding the values that the free parameters are allowed to take. Specifically a Gaussian mixture model is used to describe the mixing proportions of network weights. In the specific case where it is desirable to push small weights to zero without forcing large weights away from their required values Nowlan uses a prior *P*(*w*) which is a mixture of narrow and broad Gaussians, or

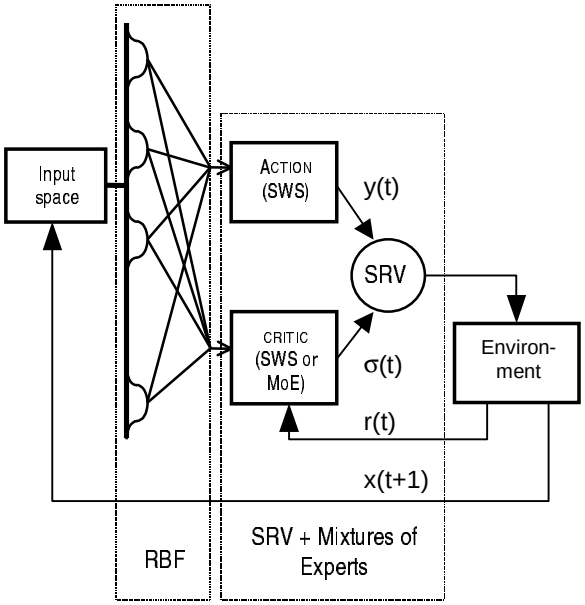

Fig 2: Network Architecture

$$p(w) = m_n \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-w^2/2\sigma_n^2\right) + m_b \frac{1}{\sqrt{2\pi}\sigma_b} \exp\left(-w^2/2\sigma_b^2\right)$$

where m_n and m_b represent the mixing proportions of the candidate Gaussians and are therefore constrained to sum to positive values with a total value of unity. This provides a framework for modelling the value weights may take in terms of the conditional probability that weight w_j was created by a specific Gaussian mixture. Naturally the concept of what denotes useful and not useful weights is application specific, hence a cost function is required which provides for the manipulation of the mixture mean and variance during learning. Such a cost function is of the form [18],

$$C = \frac{k}{\sigma_y^2}\sum_{p=1}\frac{1}{2}\left(y^{(p)} - d^{(p)}\right)^2 - \sum_{j=1}\log\left(\sum_{i=1}\pi_i p_i(w_j)\right) \qquad (4)$$

Here the objective is not to adapt weights through the soft-weight-sharing concept, but to adapt experts. The cost function is therefore reinterpreted as follows. Firstly, the mixture *prior* of (4) is equivalent to the output of the respective gating network, g_i, whereas the probability density function in the second term on the right of (4) is replaced by the maximum likelihood model of the network response, as in the case of Jordan's original formulation of the MoE cost function. This provides the required sum square cost function which, when

combined with a stochastic update rule, in this case the Solis-Wetts algorithm below, reflects both a competitive and co-operative evolution of the expert model, or in the case of the ACTION network,

$$C = \frac{1}{2}\sum_i (\hat{y} - y_i)^2 - \ln \sum_i g_i \exp\left(-\frac{(\hat{y} - y_i)^2}{2\sigma^2} \right) + \alpha \sum_i \frac{y_i^2/k}{\left(1 + y_i^2/k\right)}$$

The first term on the right represents the required square error cost with respect to expert *i*, the second is the mixture associated with expert *i*, and the third is a weight decay term. The associated weight change is therefore,

$$\frac{\partial r}{\partial y_i} = \frac{\partial r}{\partial C} \times \left[(y_i - \hat{y}) - \eta h_i \left(\frac{y_i - \hat{y}}{\sigma^2} \right) + \alpha \frac{2(y_i/k)}{\left(1 + y_i^2/k\right)^2} \right] \quad (5)$$

where α is a suitable constant, h_i is the *posterior* probability and *k* is the maximum value output *y* may take. The gate network is therefore still responsible for allocating experts to specific regions of the input space, as in the case of the original MoE definition of the learning rule.

A similar process is true for the CRITIC MoE producing a learning rule based on SWS alone. In this case the initial cost function takes the form,

$$C = \frac{1}{2}(\pi)^2 - \ln \sum_i g_i \exp\left(-(\pi)^2\right) \quad (6)$$

where π is the appropriate TD(λ) policy function [3].

Hence the CRITIC and ACTION networks are trained independently of each other and the overall system is adapted with a Solis-Wetts stochastic update process of the form,

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 neither 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.

3.3. Radial Basis Function Partitioning

The MoE paradigm provides a framework for mixing expert actions. However, in order to facilitate an environment for formulating policies in a temporal environment without recourse to recurrent interconnects, explicit partitioning of the input space is still necessary. Specifically, a Gaussian basis function is employed, where an interest lies in assessing the significance of tuning the variance term. To this end, two architectures tested: the *isotropic Gaussian* (non-generalised) and the *weighted norm* (generalised) formulation.

In the case of the non-generalised Gaussian, the normalised radial basis function, G(·), centred at t_i,

$$G\left(\|x - t_i\|^2\right) = \exp\left(-M/d^2 \|x - t_i\|^2\right)$$

where *M* is the number of centres, *d* is the maximum distance between the chosen centres and *i* takes values {1,..,*M*}. The width of all Gaussian functions are assigned using, $\sigma = d/\sqrt{2M}$ .The *generalized* RBF formulation directly incorporates the co-variance term, thus,

$$G\left(\|x - t_i\|^2\right) = \exp\left(-\tfrac{1}{2}(x - t_i)^T \Sigma_i^{-1}(x - t_i)\right)$$

with the inverse variance (covariance) matrix Σ defined by, $\frac{1}{2}\Sigma_i^{-1} = C_i^T C_i$ where C_i is a square norm diagonal weighted matrix. The effect that this parameter has is to assign a weight factor to each input condition characterizing its importance to the control action. Centre and covariance estimation occurs twice during the training cycle, section 3.4. In both cases, centres are positioned using competitive clustering with data derived by the ACTION–CRITIC interaction. Naturally, calculation of the inverse variance matrix employs the same data as the one used to position the RBF centres maintaining compatibility. To avoid any large valued parameters dominating the inverse matrix and concealing important information, the final product is normalised.

3.4. Overall training regime

The integration of the gross RBF partitioning of the input space with the MoE model takes the following form. Firstly it is noted that 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.

1. Cluster the input space using an arbitrary force generator;
2. Pre-training of the CRITIC network for the set of initial conditions;
3. Pre-training of the ACTION network for the 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.

In the following evaluation three basic contexts are considered. In the first case the MoE networks, figure 2, are replaced by MLPs. In the second case, denoted ME in the following, the CRITIC MoE is trained using the standard probabilistic cost function

(2), directly applied to the reinforcement signal *r*, (3). Training the ACTION network uses the SWS cost function, (5). In the third case, referred to as SWS in the following, the CRITIC is also trained using an SWS cost function, (6).

4. Performance Evaluation

A widely used neural network benchmark employed for the assessment of direct and indirect optimal controllers is the inverted pendulum problem [10-12, 19]. 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 ± 2.4 meters, while force must not exceed ± 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. Given the widespread knowledge of the problem we will limit our comments to specifics of the approach employed here.

In the case of this work the environment the network is subjected to takes the following form,

- No classical control policy is assumed: the network is directly applied to the environment.
- The learning to learn nothing scenario is explicitly forbidden: Adaptation only takes place when the pole position lies outside $+/- 3°$ of the vertical and +/– 0.6m of the track centre.
- The controller is required to produce a continuous control action.
- Training and test data sets are independent: as is the norm with conducting a statistical experiment, the training and test sets are unique, but sampled such that the training set does cover the region of operation; tables 1 and 2 respectively.

The training cycle of the network employs the patterns of table 1 and is completed when all satisfy the stopping criteria, section 4.1. Generalisation is assessed by examining performance over the 32 different initial conditions shown in table 2.

The following study examines three different network scenarios in order to assess the impact of *generalised* RBF, hierarchical architectures, and their contribution to the control capabilities of the neural network. Consequently, the architectures examined are: (i) an MLP network using an SRV output layer [9]; (ii) a non-hierarchical MoE network employing the *generalised* RBF; (iii) the hierarchical counterpart of (ii); and (iv) the *non-generalised* RBF counter parts of (ii) and (iii). In case (i) a single hidden layer MLP is employed in the CRITIC and ACTION components of the SRV architecture [9]. In the remaining cases, linear experts and gating networks are employed.

The second benchmark problem employed is that of reversing a truck [19-22]. This is a fundamentally different application both in terms of plant dynamics and the number of temporal steps before payoff is attained. Error criteria now take the form of the degree of deviation from the ideal docking location. Moreover, two versions are assessed for this application, direct application of the proposed architecture and a PID formulation in which the network provides estimates of the P and D terms (constant of the integral term is fixed to 0.1), but still within the context of optimal control (no *a priori* plant model). Use of such a configuration enables further partitioning of the problem into separate networks for P and D parameters. Moreover, given the long period of latency between payoff and starting condition, weight updating is only performed on the transition between RBF partitions of the input (state-) space. The typical approach to solving this problem is to either apply an indirect methodology or to employ significant amounts of *a priori* information to simplify the problem. As indicated above the interest in this work lies in assessing the ability of the network alone.

4.1. Test 1 – Pole balance

Networks are evaluated from the perspective of training requirements (CPU time), robustness to initialisation (10 different initialisation per network) and generalisation ability. In the latter case generalisation is judged from the nature of the control action, where this leads to a threshold of 30,000 simulation steps (equivalent to 10 minutes simulated time) without encountering failure. Figure 3 provides an example of such a condition, where this action is desired for all test and training conditions. Moreover, this criteria is different from that used else where, for example [12], in which the generalisation test is limited to the network lasting 1000 iterations (20 seconds) without encountering a failure condition. In this case using such a test for generalisation would result in *all* networks fulfilling the generalisation criteria.

Generalisation is summarised in tables 3 and 4 in terms of the percentage number of initialisations satisfying the above convergence criteria, and T-test of independence. From table 3 row 1, it is evident that the MLP based SRV network is unable to provide a sustainable control action. Moreover in the case of the non-generalised RBF partitioning of the input space, a significant degree of sensitivity towards the initial conditions is still observed; MoE rows 2 and 3 and hierarchical MoE 6 and 7. Furthermore, due to the larger architecture of the hierarchical MoE networks (i.e. a larger number of initialised parameters), the non-generalised hierarchical MoE case actually performs consistently worse than the non-hierarchical case. However, findings of rows 8 and 9, representing the *generalised* hierarchical versions, demonstrate sustainable control actions irrespective of the input patterns and initial conditions. Thus, the

combination of Generalised RBF partitioning and the hierarchical architecture appear to provide best-case generalisation.

Table 1: Pole balance training set

Position in meters	**Angle** in degrees	**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: Pole balance test set

Distance in meters	**Angle** in degrees	**Distance** in meters	**Angle** in degrees
+/- 1	0	+/- 0	+/- 5
	+/- 2.5	+/- 0.6	
	+/- 5.5	+/- 1.6	
	+/- 7.5	+/- 2.2	
	+/- 10		

Table 3: % Converging instances on Pole-balance.

Algorithm (all use ACTOR-CRITIC)	Test
MLP	0%
Linear Non-Generalised SWS–ME	16%
Linear Non-Generalised SWS–SWS	31%
Linear Generalised SWS–ME	70%
Linear Generalised SWS–SWS	70%
Hierarchical Non-Generalised SWS–ME	23%
Hierarchical Non-Generalised SWS–SWS	16%
Hierarchical Generalised SWS–ME	100%
Hierarchical Generalised SWS–SWS	100%

Table 4: T-test on pole-balance CPU requirements.

Algorithms Pairwise compared	Iter.	Sec.
Linear Generalised SWS–ME Vs Linear Generalised SWS–SWS	27	88
Hierarchical Non-Generalised SWS–ME Vs Hierarchical Non-Generalised SWS–SWS	3.6	5
Hierarchical Generalised SWS–ME Vs Hierarchical Generalised SWS–SWS	0	0

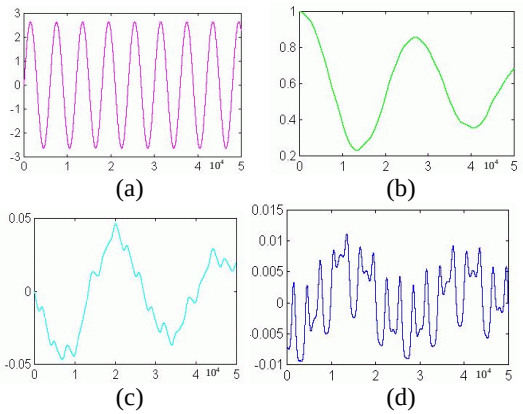

Fig 3: Example converging control action.
Key: (a) angular velocity; (b) cart position; (c) linear velocity; (d) controller force.

Training requirements are summarised in terms of the mean CPU time, figure 4, and T-test for the hypothesis that sample means are the same, table 4. In terms of the MLP network, column 1 figure 4, it appears as the least demanding, principally due to use of a gradient decent learning rule. In the case of the other architectures, remaining columns of figure 4, it appears that the characteristic of most influence is the sensitivity of the network to the initial conditions. That is to say, the non-generalised MoE networks tend to converge with a lower number of iterations, particularly in the case of the non-hierarchical case, where this is also the most sensitive to initialisation (vis-à-vis generalisation). Moreover, from the hypothesis tests of table 5 it is apparent that the cost function also has a significant effect on convergence. Cases employing the SWS cost function are significantly faster than networks trained based on the ME function.

Consideration of the aforementioned findings indicates that the square error cost function (SWS) in combination with the hierarchical MoE architecture and generalised RBF partitioning provides the same degree of generalisation and robustness at a lower level of a priori information [10-12].

4.2. Test 2 – Truck reverse

As indicated in section 4, the truck reversal problem requires credit assignment over a much longer temporal horizon. With this in mind four versions of the system are considered,

1. Unsampled – this is the hierarchical system as employed in the pole balance example;
2. Sampled – in this case weight updating of the experts only occurs when the currently active basis function changes. Thus rather than attempting to update weights at every step, *t*, the system samples the state space when basis functions detect a shift in the state space (with respect to the previous RBF condition);
3. Recurrent – the gate of the Mixtures of Expert networks (one in each layer of the hierarchy) are provided with recurrent interconnect [23];
4. PI(D) – a PID controller methodology is assumed in order to aid identification of suitable control actions over a long temporal horizon. The network, as defined above, is trained to tune the P parameter of a PI controller. The derivative component is then identified using a second ACTION network, trained using the CRITIC of the PI network in feed forward mode (provides derivative signal). This results in a pair of networks which operate in parallel tuning the PID controller.

In the latter case the PID relation if of the form,

$$u(t) = u(t-1) + K_P\{[(y(t) - y(t-1)] + K_I y(t) + K_D[y(t) - 2y(t-1) + y(t-2)]\}$$

where u is the truck steering signal; d is the distance to the target stop location; and K_P, K_D are the controller constants sort using a neural network (K_I remains fixed at 0.1 [24]).

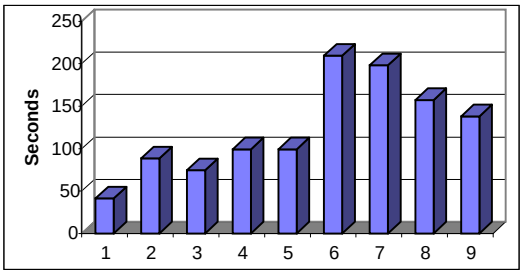

Fig 4: Mean CPU requirements to convergence
#1 MLP, **#2** Linear non-generalised ME network, **#3** Linear non-generalised SWS network **#4** Linear generalised RBF ME network, **#5** Linear generalised RBF SWS network, **#6** Hierarchical non-generalised RBF ME network, **#7** Hierarchical non-generalised RBF SWS network, **#8** Hierarchical generalised RBF ME network, **#9** Hierarchical generalised RBF SWS network

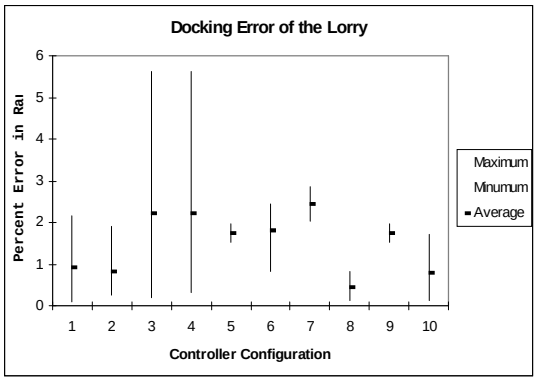

Fig 5 : Truck Controller Docking Error – Percent of total Range.
#1 : Low Level Un-sampled, #**2** : High Level Un-sampled **#3** : Low Level Sampled, #**4** : High Level Sampled, #**5** : PI network non-sampled non-recurrent, **#6** : PID equivalent of 6, #**7** : PI network sampled non-recurrent, **#8** : PID equivalent of 7, **#9** : PI network sampled recurrent, **#10** : PID equivalent of 10.

Hence, it is the responsibility of the network(s) to identify the mixing relation between proportional, integral and derivative components of the control policy [24]. Moreover the approach used here differs from previous Neural-PID controllers in which the model is explicitly limited to conditions in which there is a one-to-one relationship between input and output variables [25].

Figure 5, columns 1 and 2, describe the base-line performance of the system as directly applied, whereas columns 3 and 4 are the same system but with subsampling. Introduction of a classical PI and PID configurations, but without sampling, (columns 5 and 6), results in less sensitivity to the initial conditions, but without improving the overall accuracy of the system. Introduction of recurrency into the gate of the MoE networks (columns 7 and 8), results in a strong preference for a full PID configuration. Adding sampling of the state-space in addition to recurrency makes little further improvement.

In summary, performance of the directly applied hierarchical neural network is only bettered when a further level of partitioning is introduced, c.f. the full PID configuration, as oppose to the PI controller alone. Moreover, sub-sampling does not appear to improve performance.

5. Conclusion

An architecture for real-time control of continuous non-linear plants is proposed in which significant use of partitioning is employed. Three contributions are made,

- The significance of different cost functions is emphasised, in particular the maximum likelihood cost function typically employed by the MoE architecture provides a probabilistic framework for adapting the model, a feature which, when used in conjunction with the EM learning algorithm, provides a very efficient adaptive system. Conversely, retaining a square error cost function provides a very good regression network, where the use of annealed learning schedules helps to mitigate the effect of the increased complexity of the sum square cost function. Moreover, combined sum square – maximum likelihood cost functions provide both good regression capabilities and allow incorporation of probabilistic terminology that naturally supports the representation of measurement uncertainty and transition probabilities as frequently the case for temporal domains.
- The significance of tuning the RBF co-variance parameter is emphasised and demonstrated empirically in terms of the simulation study. It is shown that for the same set of Gaussian centres, support for co-variance terms has a major influence on the quality of the partitions identified by the following MoE framework.
- Value iteration using the TD(λ)framework is extended to incorporate generalisation both in terms of a real-valued output, SRV ACTION–CRITIC partition, and in terms of a piecewise linear decomposition of the problem into successively smaller time intervals over which the controlling function remains unchanged.

The accompanying empirical study demonstrates the ability of the architecture, to both identify solutions quickly, and to generalise the solution independent of initial conditions, whilst also supporting transparency in the solutions identified.

References

[1] Narendra K., Parasarathy K., “Identification and control of Dynamical Systems using Neural Networks”, IEEE Transactions on Neural Networks, 1(1), pp 4-27, 1990.

[2] White D.A., Jordon M.I., “Optimal Control: A Foundation for Intelligent Control”, in

Handbook of Intelligent Control, White D.A., Sofge D.A. Eds, pp 185-214, 1992.

[3] Sutton R.S., "Learning to predict by the method of temporal differences," Machine Learning, 3(1), pp 9-44, 1988.

[4] Watkins C.J.C.H, Dayan P., "Q-Learning," Machine Learning, 8(3), pp 279-293, 1992.

[5] Narendra K.S., Balakrishnan J., Ciliz M.K., "Adaptation and learning using multiple models, switching and tuning," IEEE Control Systems, 15(3), pp 37-51, 1995.

[6] Gulapalli V., "A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions," *Neural Networks*, 3, pp 671-692, 1990.

[7] Barto A.G., Sutton R.S., Anderson C.W., "Neuronlike adaptive elements that can solve difficult learning control problems," IEEE Transactions on Systems, Man and Cybernetics, 13(5), pp 834-847, 1983.

[8] Cichosz P., "Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning," Journal of Artificial Intelligence Research, 2, pp 287-318, 1995.

[9] Heywood M.I., Chan M.-C., Chatwin C.R., "Application of stochastic real-valued reinforcement neural networks to batch production rescheduling," Proceeding of the Institution of Mechanical Engineers, 221(B), pp 591-603, 1997.

[10] Lin C-,T., Lee G., "Reinforcement structure/ parameter learning for neural-network based fuzzy logic control systems," IEEE Transactions on Fuzzy Systems, 2(1), pp 46-63, 1994.

[11] Berenji H.R., Khedkar P., "Learning and tuning fuzzy logic controllers through reinforcements," IEEE Transactions on Neural Networks, 3(5), pp 724-740.

[12] Moriarty D.E., Miikkulainen R., "Efficient reinforcement learning though symbolic evolution," Machine Learning, 22, pp 11-32, 1996.

[13] Weigend A. S., Mangeas M., Srivastava A.K., "Nonlinear Gated Experts for Time Series: Discovering Regimes and avoiding Overfitting," International Journal of Neural Systems, pp 373-399, Sept. 1995.

[14] Singh S.P., "Transfer of learning by composing solutions of elemental sequential tasks," Machine Learning, 8, pp 323-339, 1992.

[15] Tham C.K., "Reinforcement learning of multiple tasks using a hierarchical CMAC architecture," Robotics and Automated Systems, 15, pp 247-274, 1995.

[16] Jordon M.I., Jacobs R.A., "Hierarchical Mixtures of Experts and the EM Algorithm," Neural Computation, 6, pp 181-214, 1994.

[17] Rao A.V., Miller D., Rose K., Gersho A., "Mixture of Experts Regression modelling by Deterministic Annealing, IEEE Transactions on Signal Processing, 45(11), pp 2811-2819, 1997.

[18] Nowlan J., Hilton G. E., "Simplifying Neural Networks by Soft Weight-Sharing" *Neural Computation*, vol. 4, pp 473-493, December 1992.

[19] Anderson C.W., Miller W.T., Challenging Control Problems, in Neural Networks for Control, Miller W.t., Sutton R.S., Werbos P.J. (Eds), MIT Press, 1990.

[20] Yamada Y, Nakashima M, shiono S, "Reinforcement Learning to Train a Cooperative Network with Both Discreate and Continuous Output Neurones" IEEE Trans on Neural Networks, IEEE, U.S.A, vol. 9, no. 6, November 1998, pp1502-1507.

[21] Kandadai R.M, Tien J.M, "A Knoweledge-Based Generating Hierarchical Fuzzy-neural Controller." IEEE Trans on Neural Networks, IEEE, U.S.A, vol. 8, no. 6, November 1997, pp1531-1541.

[22] Park Y.M, Choi M.S, Lee K.Y, "An Optimal tracking Neuro-Controller for Nonlinear Dynamic Systems." IEEE Trans on Neural Networks, IEEE, U.S.A, vol. 7, no. 5, September 1996, pp1099-1110.

[23] Cacciatore T.W., Nowlan S.J., "Mixtures of controllers for jump linear and non-linear plants," Advances in Neural Information Processing Systems, Vol 6, Cowan J., Tesauro G., Alspector J., (eds) pp 719-726, 1994.

[24] Wu Q.H., Pugh A.C., "Reinforcement learning control of unknown dynamic systems," IEE Proceedings-D, 140(5), pp 313-322, September 1993.

[25] Takagi S., Yamamoto T., Kaneda M., "A design of multivariable neuro-controllers with PID structure," 5th International Conf. N Control, Automation, Robotics and Vision, Vol 1, pp 296-300, 1998.