Neural Networks with Online Sequential Learning Ability for a Reinforcement Learning Algorithm

Hitesh Shah[1] and Madan Gopal[2]

[1] Department of Electronicsand Communication Engineering, G H Patel College of Engineering and Technology, Gujarat, India
[2] School of Engineering, Shiv Nadar University, Uttar Pradesh, India
iitd.hitesh@ gmail.com, mgopal@snu.edu.in

Abstract. Reinforcement learning (RL) algorithms that employ neural networks as function approximators have proven to be powerful tools for solving optimal control problems. However, neural network function approximators suffer from a number of problems like learning becomes difficult when the training data are given sequentially, difficult to determine structural parameters, and usually result in local minima or overfitting. In this paper, a novel on-line sequential learning evolving neural network model design for RL is proposed. We explore the use of minimal resource allocation neural network (mRAN), and develop a mRAN function approximation approach to RL systems. Potential of this approach is demonstrated through a case study. The mean square error accuracy, computational cost, and robustness properties of this scheme are compared with static structure neural networks.

Keywords: Reinforcement learning, online sequential learning, resource allocation neural network.

1 Introduction

Reinforcement learning (RL) paradigm is a computationally simple and direct approach to the adaptive optimal control of nonlinear systems [1]. In RL, the learning agent (controller) interacts with an initially unknown environment (system) by measuring states and applying actions according to its policy to maximize its cumulative rewards. Thus, RL provides a general methodology to solve complex uncertain sequential decision problems, which are very challenging in many real-world applications.

Often the environment of RL is typically formulated as a Markov Decision Process (MDP), consisting of a set of all states $\mathcal{S}$, a set of all possible actions $\mathcal{A}$, a state transition probability distribution $P:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to[0,1]$, and a reward function $R:\mathcal{S}\times\mathcal{A}\to\mathbb{R}$. When all components of the MDP are known, an optimal policy can be determined, e.g., using dynamic programming.

There has been a great deal of progress in the machine learning community on value-function based reinforcement learning methods [2]. In value-function based reinforcement learning, rather than learning a direct mapping from states to actions,

M.K. Kundu et al. (eds.), *Advanced Computing, Networking and Informatics - Volume 1*,
Smart Innovation, Systems and Technologies 27,
DOI: 10.1007/978-3-319-07353-8_11, © Springer International Publishing Switzerland 2014

the agent learns an intermediate data-structure known as a value function that maps states (or state-action pairs) to the expected long-term reward. Value-function based learning methods are appealing because the value function has well-defined semantics that enable a straight forward representation of the optimal policy, and theoretical results guaranteeing the convergence of certain methods [3], [4].

Q-learning is a common model-free value function strategy for RL [2]. *Q*-learning system maps every state-action pair to a real number, the *Q*-value, which tells how optimal that action is in that state. For small domains, this mapping can be represented explicitly by a table of *Q*-values. For large domains, this approach is simply infeasible. If, one deals with large discrete or continuous state and action spaces, it is inevitable to resort to function approximation, for two reasons: first to overcome the storage problem (*curse of dimensionality*), second to achieve data efficiency (i.e., requiring only a few observations to derive a near-optimal policy) by generalizing to unobserved states-action pairs. There is a large literature on RL algorithms using various value-function estimation techniques. Neural network function approximation (NN *Q*-learning) is one of the competent RL frameworks to deal with continuous space problem. NN generalizes among states and actions, and reduces the number of *Q*-values stored in lookup table to a set of NN weights. The back propagation NN with sigmoidal activation function can be used to learn the value function [5]. However, neural network function approximators suffer from a number of problems like learning becomes difficult when the training data are given sequentially, difficult to determine structural parameters, and usually result in local minima or over fitting. Consequently, NN function approximation can fail at finding the correct mapping from input state-action pairs to output *Q*-values.

In NN, the methods used to update the network parameters can broadly be divided into batch learning and sequential learning. In batch learning, it is assumed that the complete training data are available before training commences. The training usually involves cycling the data over a number of epochs. In sequential learning, the data arrive one by one or chunk by chunk, and the data will be discarded after the learning of the data is completed. In practical applications, new training data may arrive sequentially. In order to handle this using batch learning algorithms, one has to retrain the network all over again, resulting in a large training time.

Neural networks with on-line sequential learning ability employ a procedure that involves growing and/or pruning networks iteratively as the training data are presented. Learning is achieved through a combination of new neuron allocation and parameter adjustment of existing neurons. New neurons are added if presented training patterns fall outside the range of existing network neurons. Otherwise, the network parameters are adapted to better fit the patterns.

A significant contribution was made by Platt [6] through the development of an algorithm called resource-allocating network (RAN) that adds hidden units to the network based on the novelty of the new data in the process of sequential learning. An improved approach called RAN via extended Kalman filter (RANEKF) [7] was provided to enhance the performance of RAN by adopting an extended Kalman filter (EKF) instead of the least means square (LMS) method for adjusting network parameters. They all start with no hidden neurons and allocate the new hidden units when some criteria are satisfied. However, once the hidden neuron is generated, it will never be removed. The minimal resource allocating network (mRAN) [8, 9] is an improved version of RAN, as growing and pruning neurons are achieved with a

certain criteria based on sliding windows. All the weights and the center of each hidden neuron are tuned until certain error condition is satisfied. So, a compact network can be implemented. Other improvements of RAN developed in [10] and [11], take into the consideration the pseudo-Gaussian (PG) function and orthogonal techniques including QR factorization and singular value decomposition (SVD), and have been applied to the problem of time series analysis. In [12, 13], a growing and pruning RBF (GPA-RBF) neural network, and generalized growing and pruning RBF(GGPA-RBF) neural network approaches have been presented. The idea is to only adjust the weights of the neuron that is nearest to the most recently received input, instead of the weights of all the neurons. Significance of a neuron is measured by the average information contained in that neuron. It requires estimating the distributions and the range of input samples, as well as choosing some of the parameters appropriately before training. An online sequential extreme learning machine (OS-ELM) [14] has been proposed for learning data one-by-one and/or chunk-by-chunk with fixed or varying chunk size. However, the parameters of hidden nodes are randomly selected and only the output weights are analytically updated based on the sequentially arriving data. It should be noted that the structure in OS-ELM is fixed in advance by user.

Vamplew and Ollington in [15] compare the global (static structure such as multi-layer perceptron) versus local constructive (dynamic structure such as resource allocation network) function approximation for on-line reinforcement learning. It has been shown that the globally-constructive algorithms are less stable, but that on some tasks they can achieve similar performance to a locally-constructive algorithm, whilst producing far more compact solutions. In contrast, the RAN performed well on three benchmark problems—Acrobot, Mountain-Car, and Puddleworld, for both the on-line and off-line measures. However, this performance was only achieved by choosing parameters that allowed the RAN to create a very large number of hidden neurons.

Shiraga *et al.*. [16] proposed a neural network model with incremental learning ability for reinforcement learning task, with resource allocating network with long-term memory (RAN-LTM). From the simulation results, they showed that their proposed model could acquire more accurate action-values as compared with the following three approaches to the approximation of action–value functions: tile coding, a conventional NN model, and a version of RAN-LTM [17].

In this paper, a novel online sequential learning neural network model design for RL is proposed. We explore the use of constructive approximators (such as mRAN,an improved version of RAN) which build their structure on-line during learning/training, starting from minimal architecture. Use of mRAN We develop a mRAN function approximation approach to RL system and demonstrate its potential through a case study – two-link robot manipulator. The mean square error accuracy, computational cost and robustness properties of this scheme are compared with the scheme based on global function approximation with static structure (such as NN).

The remaining part of the paper is organized as follows: Section 2 presents architecture of mRAN and value function approximation for RL systems. Section 3 gives details of on-line sequential learning of mRAN algorithm. Section 4 compares the empirical performance on the basis of simulation results. Finally, in Section V, the conclusions are presented.

2 Approximation of Value Function Using mRAN

A minimal resource allocation network (mRAN) proposed by Yingwei *et al.* [8] is a sequential learning RBF network, which combines the growth criteria of the resource allocation network (RAN) of Platt [6] with a pruning strategy to realize a minimal network structure. It is an improvement on RAN and the RANEKF algorithm of [7] that has the ability to grow and prune the hidden neurons to ensure a parsimonious structure.

A novel value function approximator for online sequential learning on continuous state domain based on mRAN is proposed in this paper. Fig. 1(a) shows architectural view of the mRAN function approximation approach to RL (commonly known as, *Q*-learning) system.

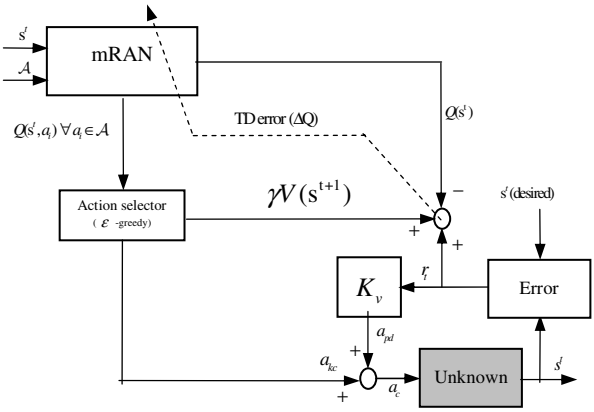

Fig. 1(a). mRAN controller architecture

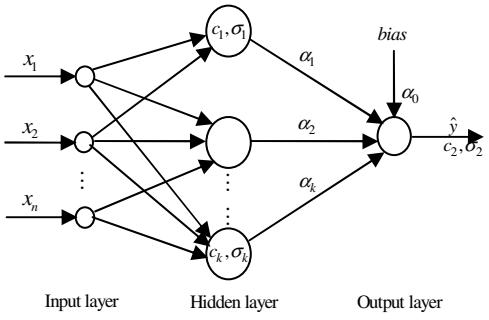

Fig. 1(b). The structure of mRAN

The state-action pair (s^t, a_t); where $\mathrm{s}^t = \{s_1^t, s_2^t, \ldots, s_n^t\} \in \mathcal{S}$ is the current system state and a_t is each possible discrete control action in action set $\mathcal{A} = \{a_i\};\ i = 1, \ldots, m$ is the input of mRAN. The output of the network (Fig. 1(b)), the estimated *Q*-value corresponding to (s^t, a_t) with K hidden neurons, has the following form:

$$Q(\mathrm{s}^t, a_t) = \hat{y}^t = f(\mathrm{x}^t) = \alpha_0 + \sum_{k=1}^{\mathrm{K}} \alpha_k \phi_k(\mathrm{x}^t) \tag{1}$$

where x is the input vector ($\mathrm{x}^t = [x_1, x_2, \ldots, x_n]^T = (\mathrm{s}^t, a_t)$) of the network, α_k is the weight connecting the k^{th} hidden neuron to the output neuron, α_0 is the bias term, and $\phi_k(\mathrm{x})$ is the response of the k^{th} hidden neuron to the input vector x, defined by the following Gaussian function:

$$\phi_k(\mathrm{x}^t) = \exp\left(-\frac{\left\| \mathrm{x}^t - \mathrm{c}_k \right\|^2}{\sigma_k^2} \right) \tag{2}$$

c_k and σ_k refer to the center and width of the k^{th} hidden neuron respectively, and $\|.\|$ indicates the Euclidean norm.

Training samples of mRAN should be obtained sequentially as the interaction between the learning agent (controller) and its environment (plant). The learning process of mRAN involves the allocation of new hidden neurons as well as adaptation of network parameters. Its learning begins with no hidden neurons. As the training samples are sequentially received, the network built based on certain growing and pruning criteria. The agent's action is selected based on the outputs of mRAN. The *temporal difference* (TD) error e^t for the output at time *t* is defined as follows:

$$e^t = \Delta Q = \eta[r_t + \gamma \max_{a \in \mathcal{A}} Q(\mathrm{s}^{t+1}, a) - Q(\mathrm{s}^t, a_t)] \tag{3}$$

It is an online sequential learning algorithm that learns an approximate state-action value function $Q(\mathrm{s}^t, a_t)$ that converges to the optimal function Q^* (commonly called *Q*-value). Online version is given by

$$Q(\mathrm{s}^t, a_t) \leftarrow Q(\mathrm{s}^t, a_t) + \Delta Q \tag{4}$$

where $\mathrm{s}^t \xrightarrow{r_t} \mathrm{s}^{t+1}$ is the state transition under the control action $a_t \in \mathcal{A}$ at instant t. In specific, control actions are selected using an exploration/exploitation policy in order to explore the set of possible actions and acquire experience through the online RL signals [2]. We use a pseudo-stochastic (ε-greedy)exploration as in [18].In ε-greedy exploration, we gradually reduce the exploration (determined by the ε parameter) according to some schedule; we have reduced ε to its 90 percent value after every 50 iterations. The lower limit of parameter ε has been fixed at 0.002 (to maintain exploration). Learning rate parameter, $\eta \in (0,1]$, can be used to optimize

the speed of learning, and $\gamma \in (0,1]$ is the discount factor which is used to control the trade-off between immediate and future reward.

3 mRAN Learning Algorithm

mRAN learning algorithm conducts the following two basic growing and pruning steps [8,9]:

Growing step: The following three conditions are checked to decide on allocation of a new hidden neuron:

$$\left\| e^t \right\| = \left\| y^t - f(\mathrm{X}^{\mathrm{t}}) \right\| > e_{\min} \tag{5}$$

$$\left\| \mathrm{X}^{\mathrm{t}} - \mathrm{c}^* \right\| > \varepsilon_t \tag{6}$$

$$e_{rms}^t = \sqrt{\frac{\sum_{i=n-(M-1)}^{n} \|e_i\|^2}{M}} > e'_{\min} \tag{7}$$

where c^* is the center of the hidden neuron which is closest to the current input X^{t}; $e_{\min}$, ε_t, and $e'_{\min}$ are thresholds to be selected appropriately. $e_{\min}$ is an instantaneous error check and is used to determine if the existing nodes are insufficient to obtain a network output. ε_t ensures that the new input is sufficiently far from the existing centers and is given by $\varepsilon_t = \max[\varepsilon_{\max}\lambda^t, \varepsilon_{\min}]$ where λ is a decay constant ($0 < \lambda < 1$). $e'_{\min}$ is used to check the root mean square error value (e_{rms}^t) of the output error over a sliding window size (*M*) before adding a hidden neuron.

If the three error criteria in the growing step are satisfied, a new hidden neuron is added to the network. The parameters associated with the new hidden neuron are assigned as follows:

$$\begin{aligned} \alpha_{k+1} &= e^t \\ c_{k+1} &= \mathrm{X}^{\mathrm{t}} \\ \sigma_{k+1} &= \mathcal{K} \left\| \mathrm{X}^{\mathrm{t}} - \mathrm{c}^* \right\| \end{aligned} \tag{8}$$

where $\mathcal{K}$ is an overlap factor (kappa), which determines the overlap of the hidden neurons responses with the hidden units in the input space.

If the observation does not meet the criteria for adding a new hidden neuron, extended Kalman filter (EKF) learning algorithm is used to adjust the network parameters w, given as:

$$\mathrm{w} = [\alpha_0, \alpha_1, \mathrm{c}_1^T, \sigma_1, \ldots, \alpha_K, \mathrm{c}_K^T, \sigma_K]^T \tag{9}$$

The update equations are given by

$$\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t-1}} + \mathrm{k}^t e^t$$

where k^t is the Kalman gain vector given by: $\mathrm{k}^t = P^{t-1}B^t[R^t + B^{t^T} P^{t-1} B^t]^{-1}$ and B^t is the gradient vector and has the following form:

$$B^t = \nabla_w \hat{y}^t = [I, \phi_1(\mathrm{X}^t)I, \phi_1(\mathrm{X}^t)\frac{2\alpha_1}{\sigma_1^2}(\mathrm{X}^t - c_1)^T, \phi_1(\mathrm{X}^t)\frac{2\alpha_1}{\sigma_1^2}\left\|\mathrm{X}^t - c_1\right\|^2, \ldots$$
$$\phi_K(\mathrm{X}^t)I, \phi_K(\mathrm{X}^t)\frac{2\alpha_K}{\sigma_K^2}(\mathrm{X}^t - c_\mathrm{K})^T, \phi_K(\mathrm{X}^t)\frac{2\alpha_K}{\sigma_K^2}\left\|\mathrm{X}^t - c_\mathrm{K}\right\|^2]^T \quad (10)$$

R^t is the variance of the measurement noise, P^t is the error covariance matrix which is updated by $P^t = [I - k^t B^{t^T}]P^{t-1} + QI$; where Q is a scalar that determines the allowed random step in the direction of gradient vector, $K(p+q+1)+p$ is size of the positive definite symmetric matrix P^t. Here $q, K,$ and p denote the number of input, hidden, and output neurons, respectively. When a new hidden neuron is added, the dimensionality of P^t is increased by

$$P^t = \begin{pmatrix} P^{t-1} & 0 \\ 0 & P^0 I \end{pmatrix} \quad (11)$$

The new rows and columns are initialized by P^0, where P^0 is an estimate of the uncertainty in the initial values assigned to the parameters. The dimension of identity matrix I is equal to the number of new parameters introduced by adding new hidden neuron.

Pruning step: To keep mRAN network to the minimal size and make sure that there are no superfluous hidden neurons existing in the network, a pruning strategy is employed.

For each observation, the normalized hidden neuron output value u_k^t should be examined to decide whether it should be removed or not. The normalized output u_k^t is expressed by the following equation:

$$u_k^t = \left\| \frac{O_k^t}{O_{\max}^t} \right\|; \; k = 1, 2, ..., K \quad (12)$$

$$O_k^t = \alpha_k \exp\left(-\frac{\left\| \mathrm{X}^t - \mathrm{c}_k \right\|^2}{\sigma_k^2} \right) \quad (13)$$

where O_k^t is the output for the k^{th} hidden neuron at time instance *t*, and $\left\| O_{\max}^t \right\|$ is the largest absolute hidden neuron output value at *t*.These normalized values are then compared with a threshold δ . If any of them falls below this threshold for *M* consecutive observations, then this particular hidden neuron is removed from the network; and the dimensionality of the covariance matrix (P^t) in the EKF learning algorithm is adjusted by removing rows and columns which are related to the pruned unit.

4 Simulation Experiments

To demonstrate the usefulness of mRAN function approximator in reinforcement learning framework, we conducted experiments using the well-known two-link robot manipulator tracking control problem.

4.1 Two-Link Robot Manipulator

We consider a two-link robot manipulator as the plant. There are two links (rigid) of lengths l_1 and l_2, and masses m_1 and m_2, respectively. The joint angles are θ_1 and θ_2, and g is the acceleration due to gravity. $\tau = [\tau_1 \;\; \tau_2]^T$ is the input torque vector, applied on the joints of the robot. Dynamic equations for a two-link robot manipulator can be represented as [19]

$$\begin{bmatrix} \alpha+\beta+2\eta\cos\theta_2 & \beta+\eta\cos\theta_2 \\ \beta+\eta\cos\theta_2 & \beta \end{bmatrix}\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}+\begin{bmatrix} -\eta(2\dot{\theta}_1\dot{\theta}_2+\dot{\theta}_2^2)\sin\theta_2 \\ \eta\dot{\theta}_1^2\sin\theta_2 \end{bmatrix}+\begin{bmatrix} \alpha e_1\cos\theta_1+\eta e_1\cos(\theta_1+\theta_2) \\ \eta e_1\cos(\theta_1+\theta_2) \end{bmatrix}+\begin{bmatrix} \tau_{1dis} \\ \tau_{2dis} \end{bmatrix}=\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \quad (14)$$

where $\tau_{dis} = [\tau_{1dis} \;\; \tau_{2dis}]^T$ is the disturbance torque vector, and $\alpha = (m_1+m_2)l_1^2$; $\beta = m_2 l_2^2$; $e_1 = g/l_1$ .Manipulator parameters are: $l_1 = l_2 = 1$ m ; $m_1 = m_2 = 1$ kg. Desired trajectory is: $\theta_{1d} = \sin(\pi t)$, and $\theta_{2d} = \cos(\pi t)$

4.2 Controller Setup

In order to setup mRAN function approximator in RL framework (mRAN controller), we used simulation parameters and learning details as follows:

System state space (continuous) has four variables, i.e., $s^t = [\theta_1 \;\; \theta_2 \;\; \dot{\theta}_1 \;\; \dot{\theta}_2]^T = [s_1 \;\; s_2 \;\; \dot{s}_1 \;\; \dot{s}_2]^T$. Each link state has two input variables: $(\theta_i^k, \dot{\theta}_i^k)$; $i = 1,2$. We define tracking error vector as: $e_r^t = \theta_d^t - \theta^t$ and cost function $r_t = \dot{e}_r^t + \Lambda e_r^t, \Lambda = \Lambda^T > 0$ with $\Lambda = \mathrm{diag}\{30,20\}$. $\theta_d^t = [\theta_{d1}^t \;\; \theta_{d2}^t]$ represents robot manipulator's desired trajectory vector. Controller action sets for link1 and link2 are $|U|(1) = [-20 \;\; 0 \;\; 20]$ Nm, and $|U|(2) = [-2 \;\; 0 \;\; 2]$ Nm, respectively. Exploration level ε decays from $0.5 \to 0.002$ over the iterations. The discount factor γ is set to 0.8; learning-rate parameter η is set to 0.2, and PD gain matrix $K_v = \mathrm{diag}\{20,20\}$. We deliberately introduce deterministic noise of ±1% of control effort with a probability of (1/3), for stochastic simulation.

4.3 Network Topologies and Learning Details

mRAN Q-learning Controller **(mRANQC):** In implementation, the mRAN has as input the state-action pair and as output, the *Q*-value corresponding to the state-action pair. In particular, the mRAN network begins with no hidden neurons. As the data are sequentially received, the network is built-up based on certain growing and pruning criteria. The input and output layers use the identity as transfer function, the hidden layer uses the Gaussian RBF as the activation function. For simplicity, the controller

uses two function approximators; one each for the two links. The initial free parameters are selected for constructing mRAN network as follows: The size of sliding data window, $M = 25$, the thresholds, $\varepsilon_{\max} = 0.8$, $\varepsilon_{\min} = 0.5$, $e_{\min} = 0.02$, $e'_{\min} = 0.002$, $\delta = 0.05$, overlap factor, $\mathcal{K} = 0.87$, the decay constant, $\lambda = 0.97$, and the estimate of the uncertainty in the initial values assigned to the parameters, $P^0 = 1.01$.

Neural Network Q-learning Controller **(NNQC):** Structurally, NNQC remains same as mRANQC; the major difference is that the NN configuration comprises of one or two hidden layers containing a set of neurons with tan-sigmoidal activation function. The number of layers and neurons depends on the complexity and the dimensions of the value-function to be approximated. We consider 18 hidden neurons. The initialization of the NN weights is done randomly, and length of training samples (l) for batch mode processing is chosen as 100.

In mRANQC (or NNQC) controller implementations, we have used controller structure with an inner PD loop. Control action to the robot manipulator is a combination of an action generated by an adaptive learning RL signal through mRAN (or NN) and a fixed gain PD controller signal. The PD loop will maintain stability until mRAN (or NN) controller learns, starting with zero initialized Q-values. The controller, thus, requires no offline learning.

5 Results and Discussion

Simulations were carried out to study the learning performance, and robustness against uncertainties, for mRAN learning approach on two-link robot manipulator control problem. MATLAB 7.10 (R2010a) has been used as simulation tool. To analyze the mRAN algorithm for computational cost, accuracy, and robustness, we compare the proposed approach with NN reinforcement learning approach.

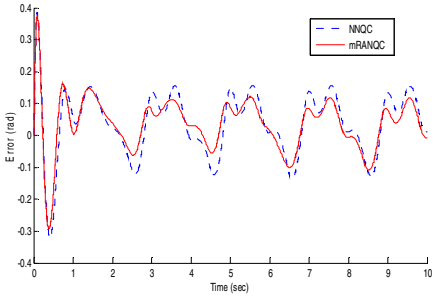

Fig. 2(a). Output tracking error (link1)

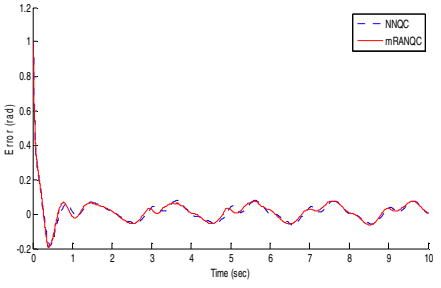

Fig. 2(b). Output tracking error (link2)

Learning performance study: The physical system has been simulated for a single run of 10 sec using fourth-order Runge-Kutta method, with fixed time step of 10 msec. Fig. 2 and Fig. 3 show the output tracking error (both the links) and control torque (both the links) for both the controllers • NNQC and mRANQC, respectively.

Table I tabulates the mean square error, absolute maximum error (max $|e(t)|$), and absolute maximum control effort (max $|\tau|$) under nominal operating conditions.

Table 1. Comparison of controllers

Controller	MSE (rad)		max $\|e(t)\|$ (rad)		max $\|\tau\|$ (Nm)		Training Time (sec)
	Link 1	Link 2	Link 1	Link 2	Link 1	Link 2	------
NNQC	0.0120	0.0065	0.1568	0.0779	83.5281	33.6758	376.78
mRANQC	0.0084	0.0065	0.1216	0.0772	81.3286	32.5861	31.55

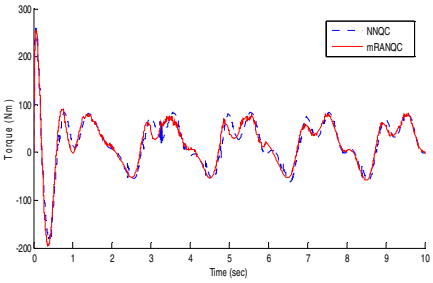

Fig. 3(a). Control torque (link1)

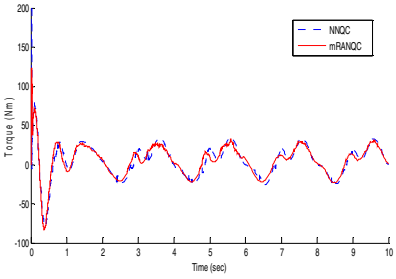

Fig. 3(b). Control torque (link2)

From the results (Fig. 2 and Fig. 3, Table 1), we observe that training time for mRANQC is lesser than NNQC. mRANQC outperforms NNQC, in terms of lower tracking errors and the low value of absolute error and control effort for both the links.

Robustness Study: In the following, we compare the performance under uncertainties of NNQC and mRANQC. For this study, we trained the controller for 20 episodes, and then evaluated the performance for two cases.

Effect of Payload Variations: The end-effector mass is varied with time, which corresponds to the robotic arm picking up and releasing payloads having different masses. The mass is varied as:

(a) $t < 2$ s ; $m_2 = 1$ kg

(b) $2 \le t < 3.5$ s ; $m_2 = 2.5$ kg

(c) $3.5 \le t < 4.5$ s ; $m_2 = 1$ kg

(d) $4.5 \le t < 6$ s ; $m_2 = 4$ kg

(e) $6 \le t < 7.5$ s ; $m_2 = 1$ kg

(f) $7.5 \le t < 9$ s ; $m_2 = 2$ kg

(g) $9 \le t < 10$ s ; $m_2 = 1$ kg .

Figs. 4(a) and (b) show the output tracking errors (both the links) and Table II tabulates the mean square error, absolute maximum error (max |$e(t)$|), and absolute maximum control effort (max |τ |) at payload variations with time.

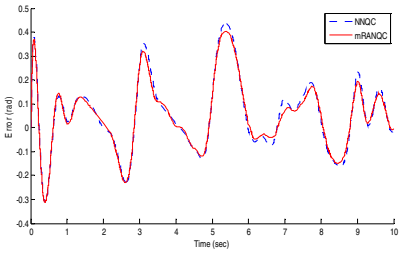

Fig. 4(a). Output tracking error (link1)

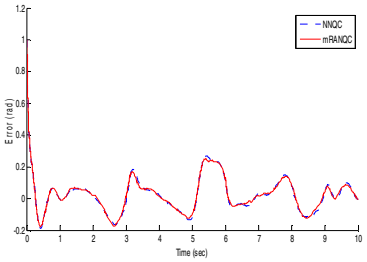

Fig. 4(b). Output tracking error (link2)

Table 2. Comparison of controllers

Controller	MSE (rad)		max |$e(t)$| (rad)		max |τ| (Nm)	
	Link 1	Link 2	Link 1	Link 2	Link 1	Link 2
NNQC	0.0261	0.0141	0.4379	0.9034	276.5605	400.3960
mRANQC	0.0227	0.0134	0.4034	0.9060	281.1594	400.4040

***Effects of External Disturbances*:** A torque disturbance τ_{dis} with a sinusoidal variation of frequency 2π rad/sec, was added with time to the model. The magnitude of torque disturbance is expressed as a percentage of control effort. The magnitude is varied as: (a) $t < 2$ s ; 0% (b) $2 \le t < 3.5$ s ; 0.2% (c) $3.5 \le t < 4.5$ s ; 0% (d) $4.5 \le t < 6$ s ; 0.8% (e) $6 \le t < 7.5$ s ; 0% (f) $7.5 \le t < 9$ s ; -0.2% (g) $9 \le t < 10$ s ; 0%. Figs. 5(a) and (b) show the output tracking errors (both the links) and Table III tabulates the mean square error, absolute maximum error (max |$e(t)$|), and absolute maximum control effort (max |τ |) for torque disturbances added with time to the model variation.

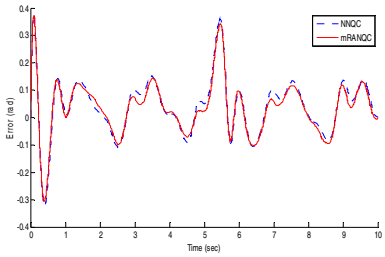

Fig. 5(a). Output tracking error (link1)

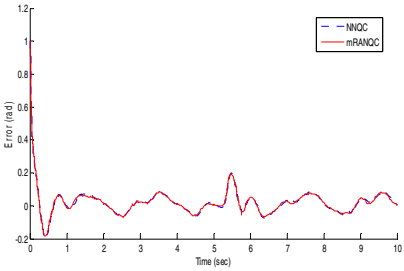

Fig. 5(b). Output tracking error (link2)

Table 3. Comparison of controllers

Controller	MSE (rad)		max $\|e(t)\|$ (rad)		max $\|\tau\|$ (Nm)	
	Link 1	Link 2	Link 1	Link 2	Link 1	Link 2
NNQC	0.0127	0.0075	0.3736	0.9078	275.4872	399.7907
mRANQC	0.0110	0.0073	0.3715	0.9062	270.0240	400.3965

Simulation results (Fig. 4 and Fig. 5, Table 2 and Table 3) shows better robustness property for mRAN based controller in comparison with NN based controller.

6 Conclusions

In order to tackle the deficiency of global function approximator (such as NN), the minimal resource allocation network (mRAN) is introduced in RL control system and a novel online sequential learning algorithm based on mRAN presented. mRAN is a sequential learning RBF network and has ability to grow and prune the hidden neurons to ensure a parsimonious structure that is well suited for real-time control applications.

From the simulation results, it is obvious that training time in mRAN based RL system is much shorter compared to the NN based RL system. This is an important feature for RL control systems from stability considerations. This feature is achieved without any loss of performance.

References

1. Sutton, R.S., Barto, A.G., Williams, R.J.: Reinforcement learning is direct adaptive optimal control. IEEE Control Syst. Mag. 12(2), 19–22
2. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge
3. Watkins CJCHLearning with delayed rewards. Ph. D. Thesis, University of Cambridge (1989)
4. Singh, S., Jaakkola, T., Littman, M., Szpesvari, C.: Convergence results for single step on-policy reinforcement learning algorithms. Machine Learning 38, 287–308 (2000)
5. Hagen, S.T., Kröse, B.: Neural Q-learning. Neural Comput. & Applic. 12, 81–88 (2003)
6. Platt, J.: A resource-allocating network for function interpolation. Neural Computation 3, 213–225 (1991)
7. Kadirkamanathan, V., Niranjan, M.: A function estimation approach to sequential learning with neural networks. Neural Computation 5, 954–975 (1993)
8. Yingwei, L., Sundararajan, N., Saratchandran, P.: A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks. Neural Computation 9, 461–478 (1997)
9. Yingwei, L., Sundararajan, N., Saratchandran, P.: Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm. IEEE Trans. on Neural Network 9, 308–318 (1998)
10. Rojas, I., Pomares, H., Bernier, J.L., Ortega, J., Pino, B., Pelayo, F.J., Prieto, A.: Time series analysis using normalized PG-RBF network with regression weights. Neurocomputing 42, 267–285 (2002)

11. Salmeron, M., Ortega, J., Puntonet, C.G., Prieto, A., Improved, R.A.N.: sequential prediction using orthogonal techniques. Neurocomputing 41, 153–172 (2001)
12. Huang, G.B., Saratchandran, P., Sundararajan, N.: An efficient sequential learning algorithm for growing and pruning RBF (GAPRBF) networks. IEEE Transcript on System Man and Cybern. B 34, 2284–2292 (2004)
13. Huang, G.B., Saratchandran, P., Sundararajan, N.: A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Transcript on Neural Network 16, 57–67 (2005)
14. Liang, N.Y., Huang, G.B., Saratchandran, P., Sundararajan, N.: A fast and accurate online sequential learning algorithm for feed forward networks. IEEE Trans. on Neural Network 17, 1411–1423 (2006)
15. Vamplew, P., Ollington, R.: Global versus local constructive function approximation for on-line reinforcement learning. In: Zhang, S., Jarvis, R.A. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 113–122. Springer, Heidelberg (2005)
16. Shiraga, N., Ozawa, S., Abe, S.: A reinforcement learning algorithm for neural networks with incremental learning ability. In: Proceeding of the 9th International Conference on Neural Information Processing, vol. 5, pp. 2566–2570 (2002)
17. Kobayashi, M., Zamani, A., Ozawa, S., Abe, S.: Reducing computations in incremental learning for feed-forward neural network with long-term memory. In: Proc. International Joint Conference on Neural Networks, pp. 1989–1994 (2001)
18. Shah, H., Gopal, M.: A fuzzy decision tree based robust Markov game controller for robot manipulators. International Journal of Automatic and Control 4(4), 417–439 (2010)
19. Green, S.J.Z.: Dynamics and trajectory tracking control of a two-link robot manipulator. J. Vibration Control 10(10), 1415–1440 (2004)