A Modular Q-Learning Architecture for Manipulator Task Decomposition

Chen K. Tham*
Department of Engineering
University of Cambridge
Cambridge CB2 1PZ
United Kingdom
ckt@eng.cam.ac.uk

Richard W. Prager
Department of Engineering
University of Cambridge
Cambridge CB2 1PZ
United Kingdom
rwp@eng.cam.ac.uk

Abstract

Compositional Q-Learning (CQ-L) (Singh 1992) is a modular approach to learning to perform composite tasks made up of several elemental tasks by reinforcement learning. Skills acquired while performing elemental tasks are also applied to solve composite tasks. Individual skills compete for the right to act and only winning skills are included in the decomposition of the composite task. We extend the original CQ-L concept in two ways: (1) a more general reward function, and (2) the agent can have more than one actuator. We use the CQ-L architecture to acquire skills for performing composite tasks with a simulated two-linked manipulator having large state and action spaces. The manipulator is a non-linear dynamical system and we require its end-effector to be at specific positions in the workspace. Fast function approximation in each of the Q-modules is achieved through the use of an array of Cerebellar Model Articulation Controller (CMAC) (Albus 1975) structures.

1 INTRODUCTION

Reinforcement learning has emerged as an important learning method for the control of autonomous agents. Its appeal lies in the fact that the agent forms its own useful behaviours with only an evaluation signal from the environment indicating how well it is doing. Many successful applications of reinforcement learning in autonomous agents have been reported by Lin (1992) and Thrun (1993); however, they usually involve learning only a single task or skill.

An autonomous agent is frequently required to perform complex tasks which can be decomposed into simpler subtasks. There are two issues in task decomposition: (1) the acquisition of an elemental skill for solving a sub-task, and (2) the coordination of several of these skills to perform the complex task.

In this paper, we describe Compositional Q-Learning (CQ-L), an approach proposed by Singh (1992) that enables the agent to learn both elemental skills and the coordination between them in composite tasks. It is efficient as it allows solutions of elemental tasks to be transferred across to composite tasks. We apply the CQ-L architecture to solve composite tasks with a simulated two-linked manipulator having state and action spaces several orders of magnitude larger than those reported in Singh (1992). In order to store Q-values in large state and action spaces as well as obtain fast function approximation, we use an array of Cerebellar Model Articulation Controller (CMAC) (Albus 1975) structures.

This paper is organized as follows. In Section 2, we describe reinforcement learning and Q-Learning. In Section 3, we describe elemental and composite tasks, the CQ-L architecture and our extensions. In Section 4, we describe the manipulator simulation and how the CQ-L architecture fits into the closed-loop system. The CMAC is described in Section 5 and details of our experiments such as the control tasks and training procedure used are given in Section 6. Results are presented in Section 7 and a comparison with related work is given in Section 8. Finally, we draw some conclusions in Section 9.

2 REINFORCEMENT LEARNING

Reinforcement learning (Barto, Sutton & Anderson 1983) is a method useful for solving Markovian sequential decision tasks where the agent operates in a stochastic dynamical environment. A model of the dynamics of the environment is not required and the performance of the agent is judged on the basis of a scalar signal known as the *reinforcement* or *payoff* which it receives from the environment. The objective is to determine the best *policy*, which specifies the agent's actions in each state, in order to maximise some cumulative measure of payoff received over time. An example is the expected *return*, which is the expected long term discounted sum of payoff. The agent performs dif-

*From January 1995: Department of Electrical Engineering, National University of Singapore. e-mail: eletck@leonis.nus.sg

ferent actions on the environment in order to discover their utilities and increases the probabilities of selecting promising actions.

Q-LEARNING

In Q-Learning (Watkins 1989), the *state-action value* $Q(x,a)$ is estimated. $Q(x,a)$ is the return in state x when action a is performed and the optimal policy is followed thereafter. The action space is discrete and a separate $Q(x,a)$ exists for each action a.

Each time the agent takes an action a from state x at time t, the current state-action value estimate for x and a denoted by $\hat{Q}_t(x,a)$ is updated as follows:

$$\hat{Q}_{t+1}(x,a) = (1-\eta)\hat{Q}_t(x,a)+\eta[R(x,a)+\gamma \max_{l\in A} \hat{Q}_t(y,l)] \quad (1)$$

where y is the actual next state, γ is the discount factor, η is a step-size parameter, A is the set of possible actions and $R(x,a)$ is the expected value of the payoff the agent will receive by taking action a in state x. The state-action value estimates for other states and actions remain unchanged.

When $\hat{Q}(x,a)$ has converged to the true state-action values $Q(x,a)$, then the *greedy* policy that selects actions according to the following criterion is optimal: $a^*(x) = \arg\max_{l\in A} Q(x,l)$. However, during learning, the agent has to explore by performing different actions. One way is to select actions according to the following probability distribution:

$$P(a \mid x) = \frac{e^{\beta \hat{Q}_t(x,a)}}{\sum_{l\in A} e^{\beta \hat{Q}_t(x,l)}} \quad (2)$$

where β is a parameter that determines the probability of selecting non-greedy actions. It is slowly increased during learning to make the policy more greedy.

3 COMPOSITIONAL Q-LEARNING (CQ-L)

A detailed description of Compositional Q-Learning can be found in Singh (1992). Here, we describe the main ideas involved and develop two extensions.[1]

3.1 ELEMENTAL AND COMPOSITE TASKS

Both elemental and composite tasks are defined for the state set S and action set A, and are subject to the same environmental dynamics with $P_{xy}(a)$ being the state transition probability from state x to state y when action a is performed. There are n elemental tasks T_i, $1 \le i \le n$, each of which cannot be broken into smaller sub-tasks. Composite task C_j is a temporal concatenation of elemental tasks i.e. $[T_1^j, T_2^j..T_{k_j}^j]$, where k_j is the number of elemental tasks in C_j. The ordering of the elemental tasks within a composite task is important.

In order for a composite task to remain a Markovian decision task, Singh extends the state representation of $x \in S$ to $x' \in S'$ by augmenting n bits to each state x so that the payoff for the composite task is a function of the extended state and not of both the state and the current elemental task. Each bit corresponds to an elemental task and is switched on when the goal state for that task is reached. The original state $x \in S$ without augmenting bits is called the *projected* state of $x' \in S'$.

Singh assumes that rewards are zero in all states except the goal states of elemental tasks. We do not make that assumption and extend the reward function such that rewards can be non-zero in states other than the desired final states of elemental tasks. Our full reward function, $r_i^f(y)$ for elemental tasks, or $r_j^{Cf}(y')$ for composite tasks, has two components. The first component, $r_i^g(y)$ for elemental task T_i, or $r_j^{Cg}(y')$ for composite task C_j, depends on the task and delivers a reward when the goal state of the task is reached. For a composite task, $r_j^{Cg}(y')$ can also give a reward when the goal state of the current elemental task in the decomposition of C_j is entered, provided the augmenting bits corresponding to elemental tasks before and including the present one in the composite task are on i.e. the elemental tasks have been performed in the correct sequence. This component enables the agent to learn different goal-directed behaviours for different tasks and is the same as the reward function used by Singh. We add a second component to the reward function: $r_i^a(y)$, which can deliver a reward in any state and depends only on the projected state y and possibly the current elemental task T_i as well, regardless of whether it is being performed on its own or as part of a composite task. Thus, the full reward function can be written as $r_i^f(y) = r_i^g(y) + r_i^a(y)$ for elemental task T_i, and $r_j^{Cf}(y') = r_j^{Cg}(y') + r_i^a(y)$ for composite task C_j when elemental task T_i in its decomposition is being performed.

The net payoff that the agent receives from the environment is influenced by stochastic effects and the cost of actions. The payoff function for elemental task T_i is $R_i(x,a) = \Sigma_{y\in S} P_{xy}(a) r_i^f(y) - c(x,a)$, where $c(x,a)$ is the cost of taking action a in projected state x. Similarly, for composite task C_j, it is $R_j^C(x',a) = \Sigma_{y'\in S'} P_{x'y'}(a) r_j^{Cf}(y') - c(x,a)$. We assume that the cost function $c(x,a)$ is task-independent[2].

When other conditions specified in Singh (1992) are satisfied, the relationship between the Q-value of composite task C_j while performing T_i (sub-task l in its decomposition) to that of elemental task T_i in the undiscounted ($\gamma = 1$) case

[1] Although we follow most of the notation used in Singh (1992), some terms are written differently in order to better describe the extensions we have introduced.

[2] In the experiments described in Section 6, we considered a deterministic environment and the cost per step c is fixed, so $R_i(x,a)$ simplifies to $r_i^f(y) - c$ and $R_j^C(x',a)$ to $r_j^{Cf}(y') - c$.

is given by :

$$Q_{T_i}^{C_j}(x', a) = Q^{T_i}(x, a) + K(C_j, l) \qquad (3)$$

where $K(C_j, l)$ is independent of state x and action a.

In the manipulator control task that we address here, rewards in states other than the goal state enable the agent to develop useful strategies for reaching its destination without collision with obstacles. It is possible to show that provided these rewards depend only on the projected state and current elemental task, as is the case with $r_i^a(y)$, the relationship in Equation 3 holds.

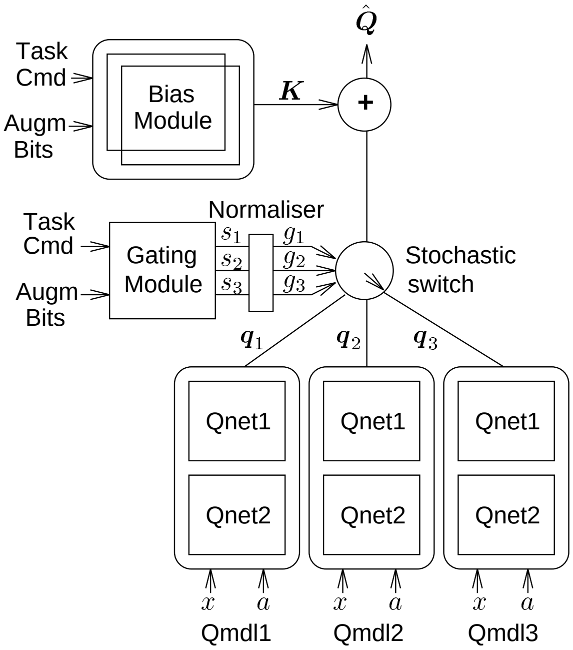


Figure 1: The CQ-L architecture for the case of an agent with three Q-Modules and two actuators; hence, each Q-module has two Q-networks. There are eleven CMAC structures within a Q-network and 66 CMAC structures in the entire architecture.

3.2 CQ-L ARCHITECTURE

The CQ-L architecture evolved out of a modular network for supervised learning consisting of expert networks and a gating network that arbitrates between them (Jacobs, Jordan, Nowlan & Hinton 1991). The basic idea here is to allow a separate expert network, which is a Q-module here, to learn each elemental task and let the gating module perform task decomposition by selecting the appropriate Q-module. A bias module compensates for the difference in the Q-values between a composite task and an elemental task in its decomposition (see Equation 3).

Our second extension to CQ-L is to the case where the agent has N actuators. The resulting architecture in which each Q-module i consists of a cluster of N Q-networks is shown in Figure 1. For example, two actions ($N = 2$) are required by the manipulator at each time-step, one for each joint, and coordination between these two actions is necessary for the successful completion of a particular task. The Q-values of a module, written as a vector, are $\boldsymbol{q}_i = [q_{i1} \; q_{i2}]^T$, where q_{i1} and q_{i2} are the outputs of the individual Q-networks within the module for a particular state and action. Grouping Q-networks in this way ensures that each Q-module is independent and enables an elemental task that requires the coordination of all actuators to be performed using a single Q-module. When Q-module i is selected, the predicted Q-values of the entire architecture is $\hat{\boldsymbol{Q}} = \boldsymbol{q}_i + \boldsymbol{K}$, where $\boldsymbol{K} = [b_1 \; b_2]^T$ is the bias module output.

The learning rules for the Q-networks and gating module that we describe now were first proposed by Nowlan (1991) in his work on competing experts and associative mixture models. We assume that the correct Q-values are generated by several regressive processes and wish to model their distributions. The output of Q-module i, $\boldsymbol{q}_i$, are the means of Gaussian density functions with variance σ^2. The output of the gating network, s_i, determines the *a priori* probability, $g_i = e^{s_i}/\Sigma_j e^{s_j}$, of the stochastic switch selecting Q-module i and depends on the inputs to the gating network, i.e. the task command and augmenting bits in this case.

Given a particular task, let us consider what happens at time t with $sel(t)$ being the currently selected Q-module. The agent is in state x and selects action a according to Equation 2, making a transition to state y. The output of the entire architecture is given by $\hat{\boldsymbol{Q}}(t) = \boldsymbol{q}_{sel(t)}(x, a) + \boldsymbol{K}(t)$, where $\boldsymbol{K}(t) = \boldsymbol{K}_{augm(t)}$ is the output of the bias module for the current task command and state of the augmenting bits. The desired output of the architecture is $\boldsymbol{D}(t) = R(t) + \hat{\boldsymbol{Q}}(t+1)$, where $R(t) = R(x, a)$ is the payoff received by the agent and $\hat{\boldsymbol{Q}}(t+1) = [\max_{l \in A} \boldsymbol{q}_{sel(t+1)}(y, l)] + \boldsymbol{K}(t+1)$.

The desired output from the Q-module section of the architecture is simply $\boldsymbol{q}_{des}(t) = \boldsymbol{D}(t) - \boldsymbol{K}(t)$, with the probability that this value is generated by Q-module i given by the Gaussian density function:

$$p(\boldsymbol{q}_{des}(t)|i) = \frac{1}{N\sigma} e^{-\frac{\|\boldsymbol{q}_{des}(t) - \boldsymbol{q}_i(t)\|^2}{2\sigma^2}} \qquad (4)$$

where $N = \sqrt{2\pi}$ is a normalizing constant. Given the desired output $\boldsymbol{q}_{des}(t)$, the *a posteriori* probability of selecting Q-module i becomes

$$p(i|\boldsymbol{q}_{des}(t)) = \frac{g_i(t) p(\boldsymbol{q}_{des}(t)|i)}{\Sigma_j g_j(t) p(\boldsymbol{q}_{des}(t)|j)} \qquad (5)$$

The log likelihood of producing the desired Q-values $\boldsymbol{q}_{des}(t)$ is $l(\boldsymbol{q}_{des}(t)) = \log \Sigma_j g_j(t) p(\boldsymbol{q}_{des}(t)|j)$. We then perform gradient ascent in the space of parameters $\boldsymbol{q}_i$ and s_i in order to maximise this log likelihood.

First, we evaluate the partial derivative of the log likelihood with respect to the output of Q-module i:

$$\frac{\partial l(\boldsymbol{q}_{des}(t))}{\partial \boldsymbol{q}_i(t)} = \frac{1}{\sigma^2} p(i|\boldsymbol{q}_{des}(t))(\boldsymbol{q}_{des}(t) - \boldsymbol{q}_i(t)) \tag{6}$$

Next, we evaluate its partial derivative with respect to the ith output of the gating network:

$$\frac{\partial l(\boldsymbol{q}_{des}(t))}{\partial s_i(t)} = p(i|\boldsymbol{q}_{des}(t)) - g_i(t)) \tag{7}$$

Finally, gradient ascent is achieved with the following:

$$\boldsymbol{q}_i(t+1) = \boldsymbol{q}_i(t) + \alpha_Q \frac{\partial l(\boldsymbol{q}_{des}(t))}{\partial \boldsymbol{q}_i(t)} \tag{8}$$

$$s_i(t+1) = s_i(t) + \alpha_g \frac{\partial l(\boldsymbol{q}_{des}(t))}{\partial s_i(t)} \tag{9}$$

where α_Q and α_g are learning rate parameters.

The bias network $\boldsymbol{K}$ is adjusted according to:

$$\boldsymbol{b}(t+1) = \boldsymbol{b}(t) + \alpha_b(\boldsymbol{D}(t) - \hat{\boldsymbol{Q}}(t)) \tag{10}$$

to absorb the difference between the Q-values of composite and elemental tasks.

In Singh (1992), the update rules in Equations 8, 9 and 10 were used at each time step to update all Q-modules, the gating module and the bias module, respectively. We found this to be unsatisfactory in the case of composite tasks and made the following changes to facilitate more robust learning: (1) the Q-modules should be updated only when the state y the system arrives in is not a goal state of an elemental task in its decomposition i.e. when the state of the augmenting bits and bias module output have not changed; (2) the bias network should be updated only when the agent arrives at a goal state of an elemental task in its decomposition. The first change prevents undesirable updates to Q-values when the outputs of the bias module have not converged to their appropriate values yet. The second change prevents Q-value prediction errors during execution of an elemental task from affecting the bias module.

In our implementation of the CQ-L architecture, we used an array of CMAC structures (described in Section 5) in each Q-module. The gating and bias networks are implemented as look-up tables since the number of tasks, augmenting bits and Q-modules is small.

4 ROBOT SIMULATION

The simulation is based on a real manipulator arm and has two revolute joints, each free to rotate through an angle of $\frac{3\pi}{2}$rads, i.e. $\frac{\pi}{4} < \theta_1 < \frac{7\pi}{4}$rads and $-\frac{3\pi}{4} < \theta_2 < \frac{3\pi}{4}$rads, where θ_1 and θ_2 are as shown in Figure 2. The manipulator is a non-linear dynamical system whose equations of motion are derived from the Lagrangian in terms of the potential and kinetic energies. These equations are integrated using a fourth-order Runge-Kutta method sampled at 500Hz.

The state $x \in S$ that is seen by the agent is made up of the angular positions and velocities of the joints i.e. $x = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$. A collision occurs when any part of the links hits an obstacle or when the manipulator attempts to move outside the allowed ranges of θ_1 and θ_2.

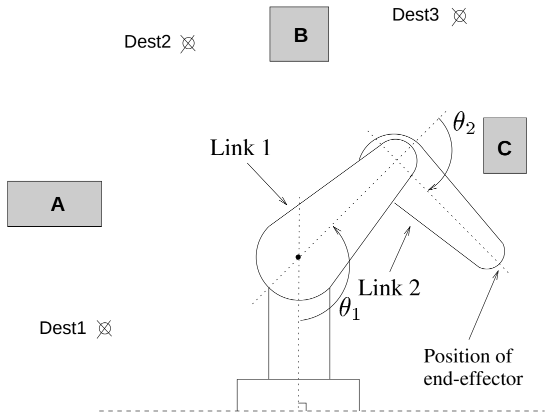


Figure 2: Robot manipulator with obstacles in the workspace. The destination of each elemental task is also shown.

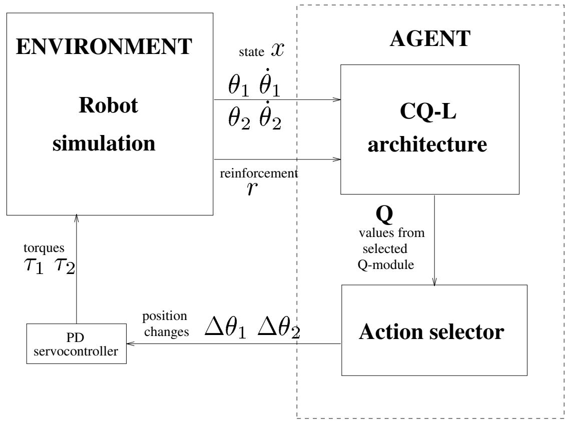


Figure 3: System block diagram showing the interaction of the agent containing the CQ-L architecture with the environment.

The way in which the CQ-L architecture fits into the entire closed-loop system is shown in Figure 3. The action selector implements the probabilistic action selection mechanism of Equation 2. The reinforcement r is $r_i^f(y)$ or $r_j^{Cf}(y')$ as described in Section 3.1. The agent controls the manipulator by generating position change commands $\Delta\theta$ to each joint. These commands, $a \in A$, are chosen from the following set of eleven possible actions: $A = \{-0.4, -0.2, -0.1, -0.05, -0.025, 0.0, 0.025, 0.05, 0.1, 0.2, 0.4\}$, in units of rads. The selected commands are then passed to a PD-servocontroller having the following form:

$$\tau = k_1 \times \Delta\theta - k_2 \times \dot{\theta} \tag{11}$$

where k_1 and k_2 are the proportional and derivative gains, respectively. This produces the torque commands for driving the manipulator. Unlike a purely kinematic system, an instantaneous response to position change commands is not possible.

5 CEREBELLAR MODEL ARTICULATION CONTROLLER (CMAC)

The CMAC (Albus 1975) is a coarse-coding structure where each region in the input space has a set of overlapping but offset hypercubes associated with it. Each hypercube is defined by quantizing functions operating on every input and it corresponds to a component of the desired output value. There are several quantizing functions for each input and the sum of the resulting components makes up the output value. The CMAC performs fast function approximation and gives good local generalization. This helps to speed up reinforcement learning since function values at a particular position in state space can be inferred from the values at neighbouring states and every point in state space does not need to be visited explicitly.

The angular positions and velocities of both joints are fed into a CMAC which is used to learn the state-action value $Q(x, a)$. In our implementation of the CQ-L architecture, each Q-network has eleven CMAC structures, one for each action a. Hence, each Q-module comprises 22 CMAC structures and there are 66 CMAC structures in the entire architecture.

Each CMAC has 12 resolution elements over the angular position input range (in Albus' notation, $Q_p = 12$) and six resolution elements over the angular velocity input range ($Q_v = 6$). There are four quantizing functions for each input ($K = 4$), two position inputs ($N_p = 2$) and two velocity inputs ($N_v = 2$), giving a total of $K(Q_p^{N_p} \times Q_v^{N_v})$ = 20,736 elements. This allows a quantizing resolution of 0.125 rads for positions and 1.0 rad/s for velocities for the input ranges that we considered, leading to 577,600 distinguishable input states.

6 EXPERIMENT DETAILS

6.1 TASKS

In all the tasks, the agent is required to drive the manipulator from an arbitrary starting arm configuration to one where its end-effector is brought to a fixed destination in the case of elemental tasks, or to several destinations, one after another, for composite tasks. The destinations are shown in Figure 2 and the elemental and composite tasks are listed in Table 1. These tasks are based on one of the experiments reported in Singh (1992, 1991) so that a direct comparison can be made.

6.2 REWARD AND COST FUNCTIONS

The learning ability of the agent depends to strength of the reinforcement signal received for different events as well as the cost per step. Since collisions occur more frequently than successful arrivals at the destination, especially in the early stages of learning, the negative reinforcement that comes with collisions cannot be too large as to overwhelm any positive reinforcement previously acquired when it reached the destination. However, insufficient negative reinforcement impairs the agent's ability to avoid obstacles.

Table 1: Elemental and Composite Tasks

Task	Description	Decomposition	$augm$ at end
T_1	reach **1**	T_1	001
T_2	reach **2**	T_2	010
T_3	reach **3**	T_3	100
C_1	reach **1, 3**	T_1T_3	101
C_2	reach **2, 3**	T_2T_3	110
C_3	reach **1, 2, 3**	$T_1T_2T_3$	111

We implemented the following reinforcement schedule [3]:

1. If the destination for elemental task T_i is reached, $r_i^g(y) = 0.5 + 0.5e^{-(|\dot{\theta}_1|+|\dot{\theta}_2|)}$. This function gives a higher than normal reinforcement if the destination is approached with low velocities. For composite task C_j, we write $r_j^{Cg}(y')$ for $r_i^g(y)$.
2. If a collision occurs, $r^a(y) = -0.025\times|\dot{\theta}|$ of the link involved, subject to the constraint that the maximum negative reinforcement is $r = -0.30$. This reward depends only on the projected state y.
3. If either $|\dot{\theta}_1|$ or $|\dot{\theta}_2|$ is greater than 10.0 rad/s, $r^a(y) = -0.05$. This indicates that excessively high velocities are undesirable and depends only on the projected state y.

The cost per step $c(x, a) = 0.025$ is constant for all states and actions.

6.3 TRAINING

Training was conducted in cycles of 5000 trials where each trial consists of at most 100 steps for elemental tasks T_1, T_2 and T_3, 250 steps for composite tasks C_1 and C_2 and 400 steps for composite task C_3. If the destination is reached or a collision occurs, the current trial ends and the next one begins with the manipulator starting from a random arm configuration.

In each trial of the corresponding experiment described in Singh (1992), the agent was presented with a randomly selected task which can be either elemental or composite, even in the early stages of learning. Even with the modifications described in Section 3.2, we could not achieve

[3] In points 2 and 3, the task subscript i has been dropped as $r^a(y)$ is also task-independent.

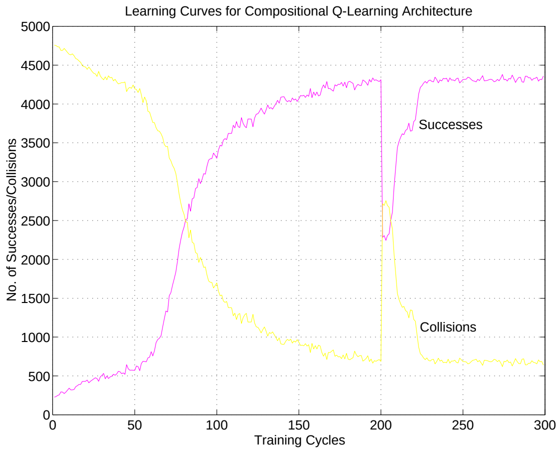


Figure 4: Performance improvement with training. In the first 200 cycles, only the three elemental tasks were presented to the agent. In the remaining 100 cycles, the three composite tasks were presented as well.

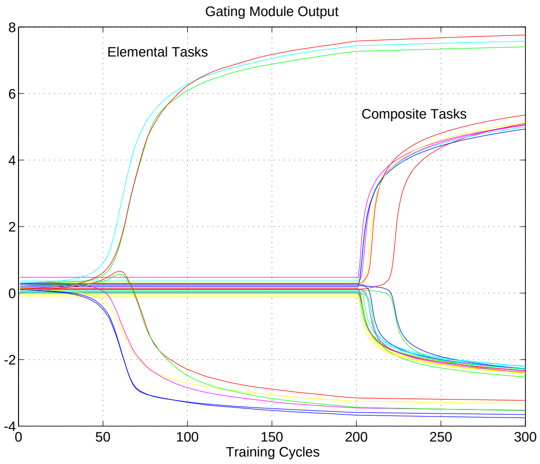


Figure 5: Variation of gating module outputs with training.

reliable learning and task decomposition if all six tasks were presented from the start. This is because the learning task which we address here is more difficult as rewards are sparser due to the larger state space and greater number of steps needed to reach a goal state.

Instead, training was done in two phases. In the first 200 training cycles, only the three elemental tasks were presented in random order. The particular Q-module to be used for each of the elemental tasks was not pre-determined and had to be selected through competition by the gating module. Then, all six tasks were presented in random order in the last 100 cycles. For composite tasks, the gating module had to learn the correct temporal concatenation of elemental tasks.

During the first 200 cycles, the Q and gating modules were updated in every time-step. However, in the last 100 cycles, the Q-modules were not adapted and updates to the gating

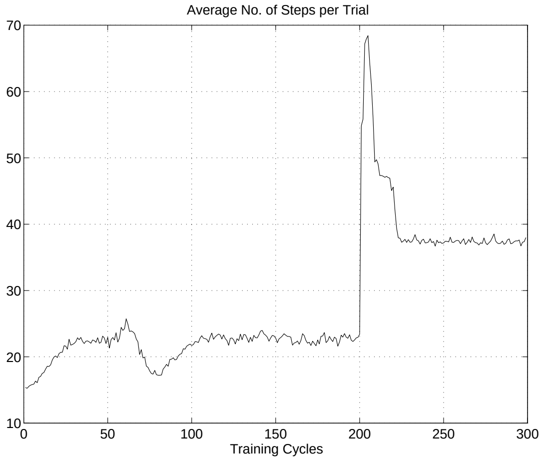


Figure 6: Variation of average number of steps per trial with training.

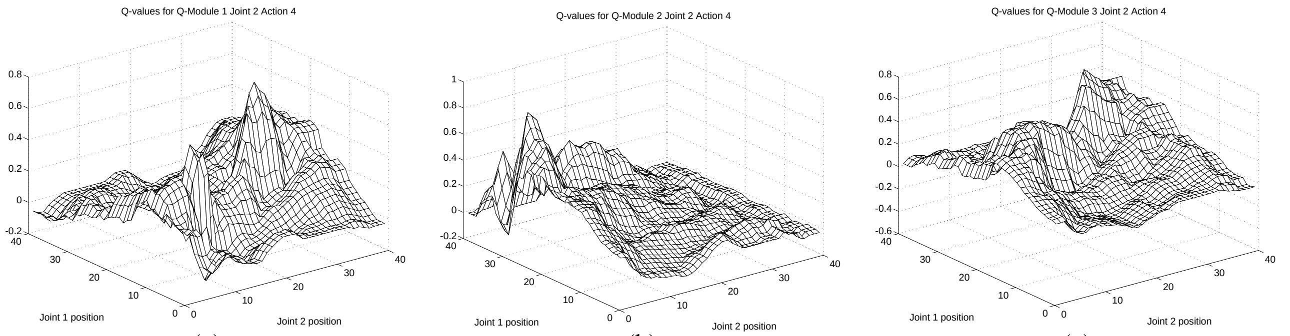
and bias modules were carried out only when the goal state of the relevant elemental task was reached, i.e. when the state of the augmenting bits had changed. In the early stages, Q-modules are selected effectively randomly during the execution of each elemental task. Thus, prediction targets are unreliable and can cause a wrong Q-module to be irreversibly favoured. Updating the gating module only when the prediction target is reliable, i.e. when the goal state of an elemental task is reached and an intermediate reward received, leads to more robust task decomposition.

7 RESULTS

Figure 4 shows how the performance of the agent improved with training in terms of the increasing number of successful arrivals at the desired destination and decreasing number of collisions in each cycle. When composite tasks were first introduced in the 200th cycle, the number of successes decreased dramatically to about half the previous level. This is because the number of elemental task requests had halved and the agent was not yet able to perform composite tasks. In the next 25 cycles, the gating module learnt how to select the appropriate Q-module for each of the elemental tasks within composite tasks, and performance improved rapidly during this time.

The way in which the outputs of the gating module s_i changed over the 300 cycles can be seen in Figure 5. By comparing corresponding stages in this figure with those in the learning curve in Figure 4, it can be seen that the number of successes increased rapidly when the outputs of the gating module for each task started to diverge and a particular Q-module was selected for each elemental task. For elemental tasks in the first 200 cycles, this happened between cycles 50 and 75; for composite tasks in the last 100 cycles, this happened between cycles 200 and 225.

Figure 6 shows the huge increase in the average number of steps per trial when composite tasks were first introduced.



(a) (b) (c)

Figure 7: Meshplots showing the variation of Q-values over the full range of manipulator movement for each of the three Q-modules for joint 2 action 4 ($\Delta\theta_2 = -0.025$rad) at fixed velocities $\dot{\theta}_1 = \dot{\theta}_2 = 0.5$rad/s.

After correct decomposition had been achieved, the average number of steps per trial is higher than before as longer trials involving composite tasks were now included in each cycle.

A snapshot of the functions learnt by the CMAC Q-modules can be seen in Figure 7. These surfaces are plotted over the entire range of angular positions for each joint for a specific action $\Delta\theta_2 = -0.025$rad and Q-network 2 (for link 2), in each of the three Q-modules. Since the input space is four-dimensional, the two angular velocities are fixed at $\dot{\theta}_1 = \dot{\theta}_2 = 0.5$rad/s. In the cases where two peaks are present on the surface, there are two inverse kinematic solutions by which the end-effector can reach the destination in the workspace. Q-modules 1, 2 and 3 have acquired the skills to perform elemental tasks T_3, T_1 and T_2 respectively.

Figure 8 shows the trajectories followed by the manipulator in order to reach the destination from different start positions in the workspace for each of the three elemental and three composite tasks.

We carried out three additional experiments to test the generality and robustness of our approach:

1. More Q-modules in the CQ-L architecture than elemental tasks: We used a CQ-L architecture with six Q-modules. The gating module selected only the three necessary Q-modules, one for each elemental task, and suppressed the contributions from the remaining three.

2. Task decomposition of other composite tasks: We presented the nine other composite tasks that can be formed from the three elemental tasks to the agent. All twelve composite tasks were decomposed within 100 cycles.

3. Noise in sensing: We added zero-mean Gaussian noise (σ_N=0.125) to the sensed angular positions and velocities. Although the number of successes in each cycle fell, the gating module was still able to decompose the tasks presented to it.

8 RELATED WORK

Mahadevan & Connell (1990) developed an architecture where multi-step behaviours were learnt using reinforcement learning and their coordination achieved by a predetermined priority ordering scheme in order to perform a box-pushing task. Maes & Brooks (1990) proposed a scheme for the coordination of behaviours in a walking robot by collecting statistics of the positive and negative feedback received when a behaviour was active and using them to determine which of several behaviours were relevant and reliable. These behaviours were hard-wired actions and did not need to be learnt. The CQ-L architecture is able to combine both the learning of multi-step elemental behaviours and the coordination between them in a single architecture.

Lin (1993*b*) studied the scaling up issues for reinforcement learning and proposed the use of neural networks trained using an experience replay algorithm to obtain generalization for faster reinforcement learning. In our approach, the fast function approximation achievable with CMAC structures removes the need for an experience replay algorithm and the storage of experiences for replay.

Lin (1993*a*) also addressed hierarchical learning where an agent learns low-level skills for solving elemental problems before learning a high-level skill, which involves the coordination of low-level skills, for solving a more complex problem. Lin treats the coordination problem as a Q-Learning task where $Q(state, skill)$ is evaluated and a skill is selected in the same way that an action is selected in low-level skills. The gating module in CQ-L does not maintain Q-values of elemental skills. It selects a Q-module according to its ability to predict Q-values in a particular task, as indicated by its posterior probability. This probability is reflected in the relative strengths of the outputs of the gating module s_i. However, the underlying principle in both cases is similar as a stochastic action selector, e.g. Equation 2, is used to select a skill or module. The selection of the most promising skill or module becomes more deterministic as learning progresses. In addition, the CQ-L

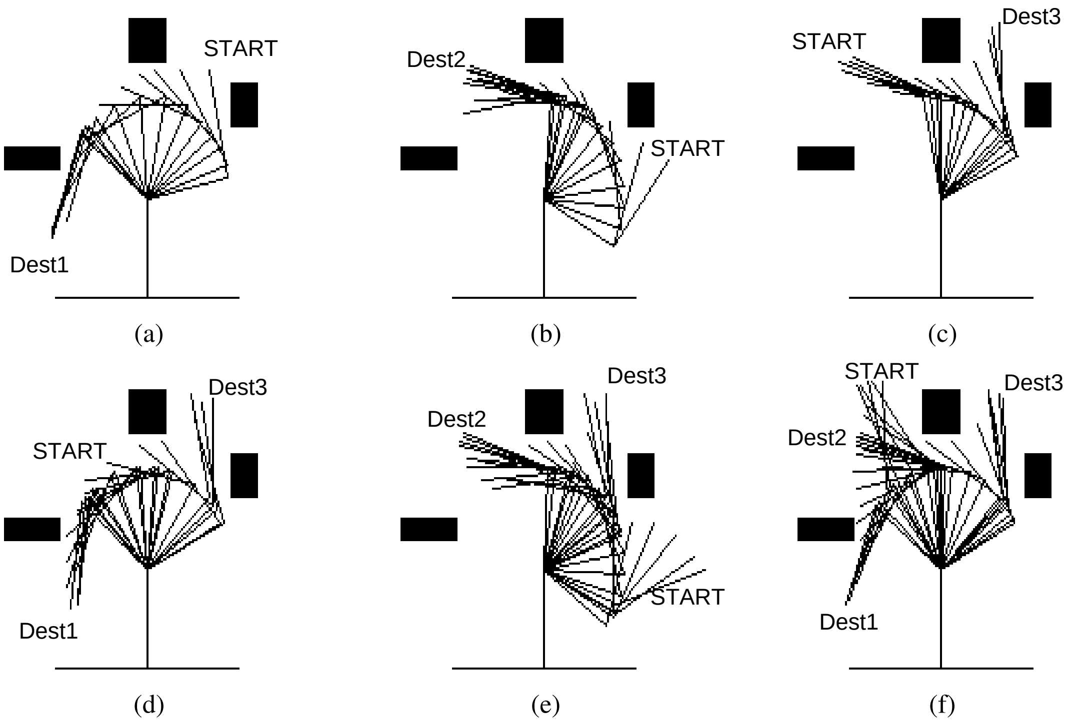


Figure 8: Trajectories followed by the robot for the three elemental tasks: T_1 (a), T_2 (b) and T_3 (c); and the three composite tasks: C_1 (d), C_2 (e) and C_3 (f).

architecture has the advantage of being able to provide the true value of $Q(x', a)$ at each stage of the composite task.

Jacobs & Jordan (1993) used a modular architecture very similar to the CQ-L architecture to model the inverse dynamics for a two-joint arm in a multi-payload robotics task. They found that the expert networks specialised in payloads with similar masses and the architecture was able to learn faster and generalize better than a single monolithic network. However, the trajectory that had to be followed by the arm was pre-specified and did not have to be learnt.

9 CONCLUSION

Singh compared the performance of CQ-L with a one-for-one architecture where a single Q-module is required to perform composite tasks on its own. Although a monolithic network can work after considerably more training for tasks with a small state space which were examined in Singh (1992), it is not feasible for tasks with a large state space and long action sequences, such as those investigated in Singh (1991) or the manipulator control tasks we addressed in this paper. A modular approach is essential for solving complex composite tasks. The CQ-L architecture is useful even when the number of elemental tasks in composite tasks is not known since the gating module is able to select only the necessary Q-modules.

Our results show that the CQ-L architecture can be successfully applied to the learning of complex composite tasks with large state and action spaces. We introduced two extensions to the original CQ-L concept: (1) a more general reward function where the agent can receive non-zero rewards in states other than the goal states of elemental tasks, and (2) the agent can have more than one actuator. In addition, we made several modifications to the strategy for parameter updates in order to achieve reliable task decomposition. Fast function approximation by CMAC structures enabled local generalization to be obtained without requiring experience replay.

As mentioned above, the CQ-L architecture is very similar to modular networks (Jacobs et al. 1991) for supervised learning. Recent developments such as the use the Expectation-Maximization algorithm for parameter updates in the Hierarchical Mixtures of Experts architecture (Jordan & Jacobs 1993) have reduced significantly the training time required to reach a given level of performance in supervised learning tasks. It will be interesting to see how well these methods work in reinforcement learning situations.

Acknowledgements

We are grateful to Satinder Singh at MIT, and our colleagues at CUED: Tim Jervis, Steve Waterhouse, Gavin Rummery and Andrew Senior, for their comments. Chen K. Tham is supported by an Overseas Graduate Scholarship from the National University of Singapore.

References

Albus, J. (1975), 'Data storage in the cerebellar model articulation controller (CMAC)', *Journal of Dynamic Systems, Measurement and Control* **97**(3), 228–233.

Barto, A., Sutton, R. & Anderson, C. (1983), 'Neuronlike elements that can solve difficult learning control problems', *IEEE Transactions on Systems, Man and Cybernetics* **SMC-13**(5), 835–846.

Jacobs, R. & Jordan, M. (1993), 'Learning piecewise control strategies in a modular neural network architecture', *IEEE Transactions on Systems, Man and Cybernetics* **23**(2), 337–345.

Jacobs, R., Jordan, M., Nowlan, S. & Hinton, G. (1991), 'Adaptive mixtures of local experts', *Neural Computation* **3**, 79–87.

Jordan, M. & Jacobs, R. (1993), Hierarchical mixtures of experts and the EM algorithm, Technical Report 9301, MIT Computational Cognitive Science.

Lin, L. (1992), 'Self-improving reactive agents based on reinforcement learning, planning and teaching', *Machine Learning* **8**(3/4), 293–321.

Lin, L. (1993*a*), Hierarchical learning of robot skills by reinforcement, *in* 'Proceedings of the International Conference on Neural Netwroks (ICNN'93)', Vol. 1, pp. 181–186.

Lin, L. (1993*b*), Scaling up reinforcement learning for robot control, *in* 'Machine Learning: Proceedings of the Tenth International Conference (ML93)', Morgan Kaufmann.

Maes, P. & Brooks, R. (1990), Learning to coordinate behaviours, *in* 'Proceedings of the 8th AAAI Conference', Morgan Kaufmann, pp. 796–802.

Mahadevan, S. & Connell, J. (1990), Automatic programming of behaviour-based robots using reinforcement learning, Research Report RC 16359 # 72625, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

Nowlan, S. (1991), Soft Competitive Adaptation: Neural Network Learning Algorithms based on Fitting Statistical Mixtures, PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Singh, S. (1991), The efficient learning of multiple task sequences, *in* J. Moody, S. Hanson & R. Lippman, eds, 'Advances in Neural Information Processing Systems 4', Morgan Kaufmann, San Mateo, CA, pp. 251–258.

Singh, S. (1992), 'Transfer of learning by composing solutions of elemental sequential tasks', *Machine Learning* **8**(3/4), 323–339.

Thrun, S. (1993), Exploration and model building in mobile robot domains, *in* 'Proceedings of the IEEE International Conference on Neural Networks', San Francisco, CA.

Watkins, C. (1989), Learning from Delayed Rewards, PhD thesis, University of Cambridge, Cambridge, UK.