CHAPTER 5

MULTI–PAYLOAD ROBOTICS TASK

The previous chapter showed that the modular architecture's rate of learning may be relatively unaffected by the inconsistent training information that characterizes temporal crosstalk. This chapter shows that this robustness leads to superior performance on at least one interesting "real world" task. This task is to control a simulated robot arm to move a variety of payloads, each of a different mass, along a specified trajectory. Two important properties of the modular architecture are demonstrated in this chapter. A suitably designed modular architecture tends to allocate the same expert network to similar tasks, resulting in the benefits of positive transfer of training, and different expert networks to dissimilar tasks, thereby avoiding the detrimental effects of negative transfer of training. In addition, the modular architecture can be easily modified so as to learn to perform a family of tasks by using one network to learn a shared strategy that is used in all contexts along with other networks that learn modifications to this strategy that are applied in a context sensitive manner.

The chapter is organized as follows. Section 5.1 describes the procedure used to train several connectionist systems to perform the multi–payload robotics task. Section 5.2 compares the results of training these systems. Section 5.3 presents a demonstration of the ease with which a priori knowledge of the domain may be embedded in the modular architecture.

5.1 Training Procedure

Frequently, controllers of robot arms employ fixed models of the dynamics of the arm. Unfortunately, the use of such models has several drawbacks. For many robot systems, accurate models are hard to formulate and model parameters are difficult to determine. Additionally, such models may be inflexible in the sense that they are closely dependent on the parameters of the system being modeled and require significant modification if the system is altered or interacts with a new tool or payload. Consequently, many researchers study techniques to allow controllers to learn desired control laws without detailed knowledge of the robot system.

One such technique uses an adaptive feedforward controller and a fixed feedback controller to control a robot arm. The feedback controller aids in generating training data that the feedforward controller uses to learn a model of the arm's inverse dynamics. To the best of our knowledge, this training procedure was developed independently by Atkeson and his colleagues (Atkeson and McIntyre [2], Atkeson and Reinkensmeyer [3]), Kawato and his colleagues (Kawato [37], Kawato, Furukawa, and

Suzuki [38]), and Miller and his colleagues (Miller [46], Miller, Glanz, and Kraft [47]). The connectionist systems reported in this chapter were trained using this procedure and, thus, it is now described in detail.

Let the state of a robot arm at time t be represented by its joint positions, $\theta(t)$, and joint velocities, $\dot{\theta}(t)$. In order to achieve joint accelerations $\ddot{\theta}(t)$, torques $\tau(t)$ must be applied to the arm. This relationship is known as the inverse dynamics of the arm and is written:

$$\tau(t) = f^{-1}(\theta(t), \dot{\theta}(t), \ddot{\theta}(t)). \tag{5.1}$$

The goal of the procedure is to train a feedforward controller to accurately model this relationship.

Each time step of the procedure contains a performance stage and a training stage. During the performance stage, both a feedforward controller and a feedback controller compute torques used to control the arm (see Panel A of Figure 5.1). The inputs to the feedforward controller are the desired joint positions, velocities, and accelerations for the current time step as specified by the desired trajectory. The outputs are the feedforward torques. The inputs to the feedback controller are the desired and actual joint positions and velocities. The outputs are the feedback torques. The sum of the feedforward and feedback torques are applied to the arm and the resulting joint accelerations are observed. During the training stage, the feedforward controller receives new inputs, namely the actual joint positions, velocities, and accelerations, and computes new outputs (see Panel B of Figure 5.1). The desired outputs of this

controller are the actual torques applied to the arm. In this manner, the feedforward controller learns a model of the arm's inverse dynamics. During the initial time steps of the procedure, most of the torques are generated by the feedback controller and the arm imprecisely follows the desired trajectory. However, after the feedforward controller learns an accurate model of the arm's inverse dynamics, it generates most of the torques and the arm faithfully follows the desired trajectory.

Our interest is in training connectionist systems to serve as feedforward controllers for a robot arm when a variety of payloads, each of a different mass, must be moved along a specified trajectory. In comparison with the case where the payload is fixed, controlling an arm with a time–varying payload is more difficult. Assuming that the payload is modeled as part of the last link of the arm, this difficulty arises because the inverse dynamics of the arm are strongly dependent on the mass of the payload. [1] In our simulations, the multi–payload task is particularly difficult because we assumed that the connectionist systems could detect the identity of a payload (e.g., payload A or payload B) but not the mass of the payload. Six payloads were used with masses of 0, 2, 10, 15, 22, and 27 kg respectively.

One complete simulation of the training of a connectionist system is called a run and is described using four time frames. Each run was divided into ten epochs. Each epoch was divided into six bins corresponding to the six payloads. At the start

[1]The inverse dynamics of the arm are also dependent on the moments of inertia and center of mass of the payload. However, as mentioned below, we modeled the links of the robot arm and the payloads as point masses and, thus, these factors did not play a role in our simulations.

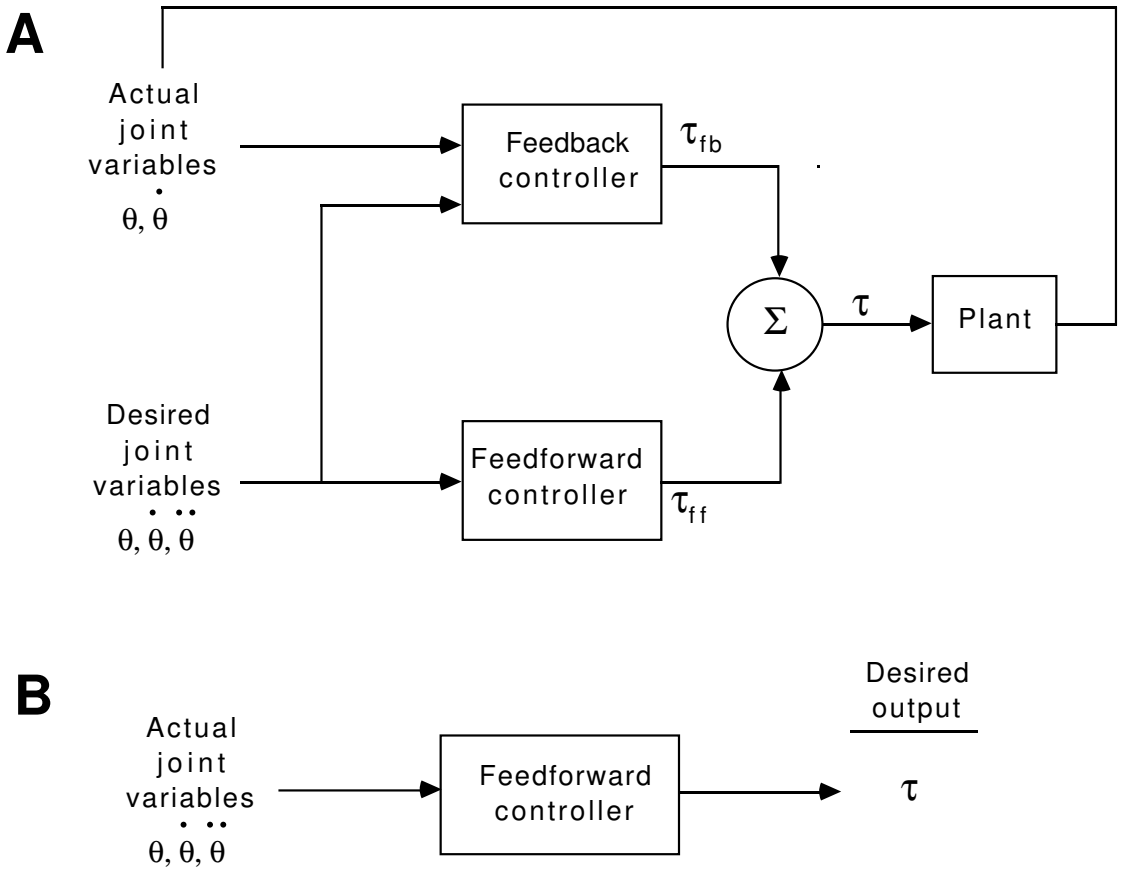

Figure 5.1: A: During the performance stage, both a feedforward controller and feedback controller compute torques used to control the robot arm. B: During the training stage, the feedforward controller receives the actual joint variables. Its desired output is the torque applied to the arm.

of each bin, a payload was randomly selected from a uniform distribution such that each payload was selected exactly once in each epoch. During a bin, the connectionist system attempted to direct the robot arm with the selected payload along the desired trajectory five times. Torques were applied at a rate of 100 Hz and the desired movement has a duration of 2 seconds. Therefore, each traversal of the trajectory lasted 200 time steps. In summary, a run comprised 10 epochs; an epoch comprised 6 bins; a bin comprised 5 traversals of the desired trajectory; a traversal comprised 200 time steps.

The robot arm is the two–joint planar manipulator shown in Figure 5.2. Each link of the arm was modeled as a point mass located at its distal end. The payload was modeled as part of the second link. The dynamic equations were obtained from Craig [15]. These equations were integrated using a fourth–order Runge–Kutta method with a time step of 0.01 seconds. The parameters of the arm are listed in Table 5.1. The feedback controller was a position controller with a gain of 1000 kg $\cdot$ m^2/s^2/radian. The parameters of the arm and of the feedback controller were obtained from Miller, Glanz, and Kraft [47]. The desired trajectory is a straight line horizontal movement with a velocity of zero at the endpoints and the maximum velocity at the center of the trajectory. If the origin of a Cartesian coordinate system lies at the axis of joint 1, then the coordinates of the desired trajectory, quantized in meters, are

$$x(t) = -\cos(\frac{t\,\pi}{2}) \tag{5.2}$$

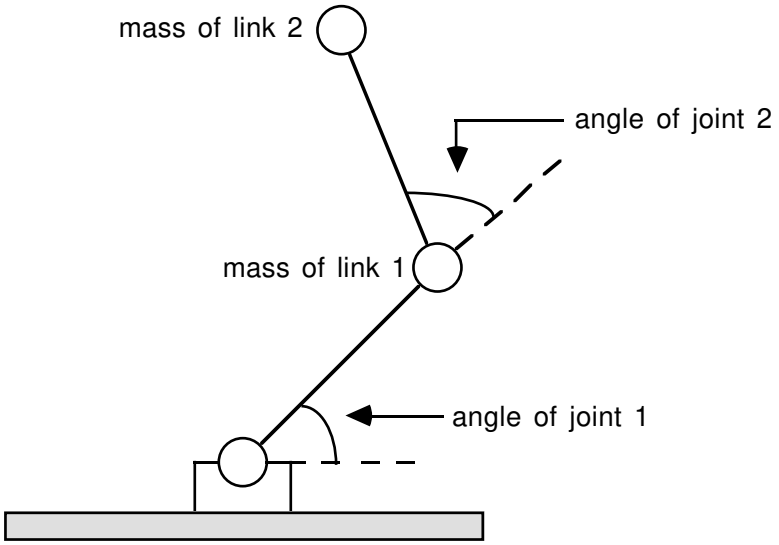

Figure 5.2: Two–joint planar arm.

and

$$y(t) = 1, \tag{5.3}$$

where $t \in [0, 2]$.

Table 5.1: Parameters of the robot arm.

Parameter	Link 1	Link 2
Length	1.0 m	0.8 m
Mass	10 kg	10 kg
Viscous friction coefficient	30 kg $\cdot$ m^2/s/radian	20 kg $\cdot$ m^2/s/radian

5.2 Comparison of Architectures

Six connectionist systems were trained to serve as feedforward controllers. These six systems consist of two systems from each of three classes of architectures. The two systems from each class were trained using different sets of input vectors. These sets of input vectors are labeled L and NL (L stands for linear; NL stands for nonlinear). The input vectors in both sets contain 19 components. Ten components are binary and represent ten possible payloads (only six of the payloads were actually used in the simulations). The remaining nine components are real–valued and represent the robot arm's joint positions, velocities, and accelerations. It is in the representation of the robot arm's joint variables that the two sets of input vectors differ. In the input vectors of set L, the nine components are the transformations (labeled $f_i(\theta, \dot{\theta}, \ddot{\theta})$) of the joint variables shown on the left of Table 5.2. These transformations were conveniently chosen so as to facilitate learning of the robot arm's inverse dynamics. More precisely, these transformations were selected so that, for a fixed payload, the inverse dynamics of the robot arm are linear in the transformations (hence, the input vectors using these transformations are members of set L). This practice was adopted from Kawato et al. [38] who argued that such useful transformations can be prepared despite very incomplete knowledge of the arm's dynamics. In the input vectors of set NL, the nine components are the transformations (labeled $g_i(\theta, \dot{\theta}, \ddot{\theta})$) of the joint variables shown on the right of Table 5.2. Even if the payload is fixed, the inverse

Table 5.2: f_i and g_i are the transformations of the robot arm's joint positions, velocities, and accelerations used in the input vectors of sets L and NL respectively. The subscripts on $\theta, \dot{\theta}$, and $\ddot{\theta}$ are the joint number.

Set L					Set NL			
i	$f_i(\theta, \dot{\theta}, \ddot{\theta})$	i	$f_i(\theta, \dot{\theta}, \ddot{\theta})$		i	$g_i(\theta, \dot{\theta}, \ddot{\theta})$	i	$g_i(\theta, \dot{\theta}, \ddot{\theta})$
1	$\ddot{\theta}_1$	6	$\dot{\theta}_2^2 \sin\theta_2$		1	θ_1	6	$\ddot{\theta}_2$
2	$\ddot{\theta}_2$	7	$\dot{\theta}_1 \dot{\theta}_2 \sin\theta_2$		2	θ_2	7	$\cos\theta_1$
3	$\ddot{\theta}_1 \cos\theta_2$	8	$\cos\theta_1$		3	$\dot{\theta}_1$	8	$\cos\theta_2$
4	$\ddot{\theta}_2 \cos\theta_2$	9	$\cos(\theta_1 + \theta_2)$		4	$\dot{\theta}_2$	9	$\cos(\theta_1 + \theta_2)$
5	$\dot{\theta}_1^2 \sin\theta_2$				5	$\ddot{\theta}_1$		

dynamics of the robot arm are nonlinear in these transformations (hence, the input vectors using these transformations are members of set NL).

The specifications of the six connectionist systems trained to serve as feedforward controllers are listed in Table 5.3. The six systems are grouped into three classes of architectures. For each system, the system label indicates the class of architectures to which the system belongs and the set of input vectors used in training the system. Systems N–L and N–NL are single networks (N stands for network). System N–L was trained with the input vectors from set L and system N–NL was trained with the input vectors from set NL. Systems MA–L and MA–NL are modular architectures with three expert networks and one gating network. Systems MAS–L and MAS–NL are referred to as *modular architectures with share networks*. This class of architectures

is formed by modifying the standard modular architecture, and its use requires some explanation.

Throughout this thesis, the importance of learning to allocate different networks to learn different tasks has been emphasized. However, in some circumstances, this strategy has an important drawback. Consider two tasks that possess many similarities but also some dissimilarities. Since the tasks possess similarities, the features learned about one task may also be useful for learning and performing the second task. Therefore, allocating distinct networks to learn each task is a poor strategy. Alternatively, since the tasks possess dissimilarities, a network trained to perform both tasks may learn slowly and develop a relatively uninterpretable representation. Therefore, allocating one network to learn both tasks is a poor strategy.

An ideal strategy may be to allocate one network to learn the features that are unique to the first task, one network to learn the features that are unique to the second task, and one network to learn the features that are common to the two tasks. When performing a task, a system must combine the outputs of the network that learns the common features with the outputs of the network that learns the features particular to the task. In an attempt to design a modular architecture that can discover this ideal strategy, we have modified the standard modular architecture. This modified modular architecture, called a modular architecture with a share network, was inspired by an architecture that was proposed, but never implemented, by Kawato et al. [38].

Table 5.3: Single networks, modular architectures, and modular architectures with share networks used in the multi–payload robotics experiments.

Single Networks			
System	Network	Input	Output
N–L	$19 \rightarrow 10 \rightarrow 2$	payload identity and $f_i(\theta, \dot{\theta}, \ddot{\theta})$	feedforward torques
N–NL	$19 \rightarrow 10 \rightarrow 2$	payload identity and $g_i(\theta, \dot{\theta}, \ddot{\theta})$	feedforward torques

Modular Architectures				
System	Expert Networks	Expert Networks' Input	Gating Network	Gating Network's Input
MA–L	$9 \rightarrow 2$	$f_i(\theta, \dot{\theta}, \ddot{\theta})$	$10 \rightarrow 3$	payload
	$9 \rightarrow 2$	$f_i(\theta, \dot{\theta}, \ddot{\theta})$		identity
	$9 \rightarrow 2$	$f_i(\theta, \dot{\theta}, \ddot{\theta})$		
MA–NL	$9 \rightarrow 10 \rightarrow 2$	$g_i(\theta, \dot{\theta}, \ddot{\theta})$	$10 \rightarrow 3$	payload
	$9 \rightarrow 10 \rightarrow 2$	$g_i(\theta, \dot{\theta}, \ddot{\theta})$		identity
	$9 \rightarrow 10 \rightarrow 2$	$g_i(\theta, \dot{\theta}, \ddot{\theta})$		

Modular Architectures With Share Networks						
System	Share Network	Share Network's Input	Expert Networks	Expert Networks' Input	Gating Network	Gating Network's Input
MAS–L	$9 \rightarrow 2$	$f_i(\theta, \dot{\theta}, \ddot{\theta})$	$0 \rightarrow 2$	none	$10 \rightarrow 3$	payload
			$9 \rightarrow 2$	$f_i(\theta, \dot{\theta}, \ddot{\theta})$		identity
			$9 \rightarrow 2$	$f_i(\theta, \dot{\theta}, \ddot{\theta})$		
MAS–NL	$9 \rightarrow 10 \rightarrow 2$	$g_i(\theta, \dot{\theta}, \ddot{\theta})$	$0 \rightarrow 2$	none	$10 \rightarrow 3$	payload
			$9 \rightarrow 10 \rightarrow 2$	$g_i(\theta, \dot{\theta}, \ddot{\theta})$		identity
			$9 \rightarrow 10 \rightarrow 2$	$g_i(\theta, \dot{\theta}, \ddot{\theta})$		

The modular architecture with a share network, which is illustrated in Figure 5.3, consists of a share network as well as a set of expert networks and a gating network. During training, the share network learns a strategy that is useful for performing all tasks. This strategy is referred to as a shared strategy. The expert networks learn modifications to the shared strategy that are particular to individual tasks. The output of the architecture, $\mathbf{y}$, equals the output of the share network, $\mathbf{y}_s$, plus the gated outputs of the expert networks:

$$\mathbf{y} = \mathbf{y}_s + \sum_{i=1}^{n} g_i \mathbf{y}_i. \tag{5.4}$$

The training procedure for this architecture is nearly identical to the training procedure for the standard modular architecture. The weights of the share network, like the weights of the expert networks, are modified so to reduce the sum of squared error between the desired and actual outputs of the architecture (Equation 3.2). The training of the gating network is slightly different in the two varieties of modular architectures. Recall that for the standard modular architecture, the winner of the competition among the expert networks is defined as the network whose output most closely approximates the desired output of the system. For the modular architecture with a share network, the winning expert network is defined as the network whose output most closely approximates the difference between the desired output of the system and the output of the share network. Define the error for expert network i to be

$$J_{\mathbf{y}_i} = \frac{1}{2}(\mathbf{y}^* - \mathbf{y}_s - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_s - \mathbf{y}_i). \tag{5.5}$$

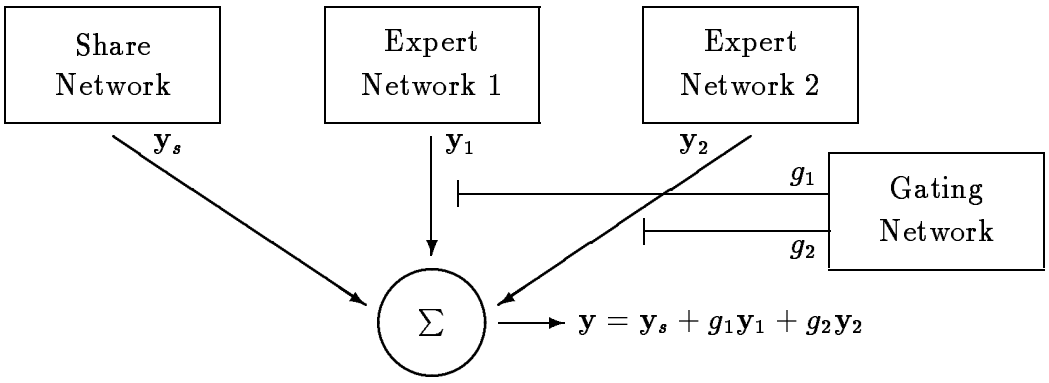

Figure 5.3: A modular architecture with a share network.

The expert network with the smallest error is the winner.

In regard to the problem of controlling a robot arm to move a variety of payloads along a specified trajectory, there are many possible decompositions of this problem into a shared strategy and a set of modifications to this strategy. The decomposition that we are interested in defines the shared strategy to be the application of the correct feedforward torques to control the arm with no payload along the desired trajectory. The modifications to this strategy are to add the extra torques required to compensate for the mass of the payload.

Systems MAS–L and MAS–NL have one share network, three expert networks, and one gating network (see Table 5.3). In each system, the first expert network doesn't contain any input units and contains two output units whose activations are always zero. For this reason, this expert network is referred to as the null expert network. When designing these systems, it was hoped that the systems are suitably structured so that the share network and the null expert network would learn to control the

arm with no payload, the share network and another expert network would learn to control the arm with a light payload, and the share network and the remaining expert network would learn to control the arm with a heavy payload. The null expert network is needed because we want the share network alone to control the arm with no payload, but the gating network's error function (Equation 3.6) specifies that, in response to each input pattern, one of the expert networks should be "switched on" and all other expert networks should be "switched off". Therefore, we provided the null expert network in the hope that this expert network would be "switched on" in the absence of a payload and, thus, the share network alone would learn to control the arm in this situation.

The learning curves for the systems trained with the input vectors from set L are shown in Figure 5.4. The horizontal axis gives the number of epochs. The vertical axis gives the joint root mean squared error (RMSE) in radians averaged over 25 runs. [2] For many parameters of each system (e.g., step size and momentum), we used the values that appeared to give the best performance. These values and

[2]This joint RMSE was computed as follows. At each time step, the squared differences between the desired and actual joint positions was determined. The joint RMSE was determined for each traversal of the trajectory by averaging the squared differences over all 200 time steps of the traversal, and then taking the square root of this average. The joint RMSE for each epoch was determined by averaging the joint RMSE of the 30 traversals (6 payloads times 5 traversals per payload) that occurred in each epoch. The joint RMSE shown in Figure 5.4 was determined by averaging the joint RMSE of each epoch over the 25 runs.

additional details of the simulations are provided in Appendix B. Clearly, the two modular architectures MA–L and MAS–L performed better than the single network N–L. This superior performance is due to the fact that the multi–payload robotics task is characterized by a high degree of temporal crosstalk. Each system attempted to model the inverse dynamics of the robot arm with a payload that is fixed within each bin of 1000 consecutive time steps, but varied between bins. As shown in Chapter 4, the rate of learning of single networks is often diminished by temporal crosstalk, whereas modular architectures tend to be robust in the presence of temporal crosstalk. The performances of the two modular architectures were approximately equal.

Figures 5.5 and 5.6 show how the modular architecture MA–L and the modular architecture with a share network MAS–L allocated their expert networks to the various payloads. Corresponding to each allocation is a bar, and the horizontal axis numbers the different allocations. For example, 25 runs of system MA–L resulted in 5 different allocations of expert networks to payloads and, thus, there are five different bars in the bar graph shown in Figure 5.5. The allocation that each bar corresponds to is given by the arrangement of the six numbers above the bar where the numbers are the masses in kilograms of the six payloads. Payloads that were allocated to the same expert network have their masses bracketed together. The vertical axis gives the number of runs in which an architecture allocated its expert networks in the corresponding manner. For example, the leftmost bar of Figure 5.5 shows that on 12 of the 25 runs, one expert network of system MA–L won the competition to control the arm when the payload was 0 or 2 kg, another expert network won the

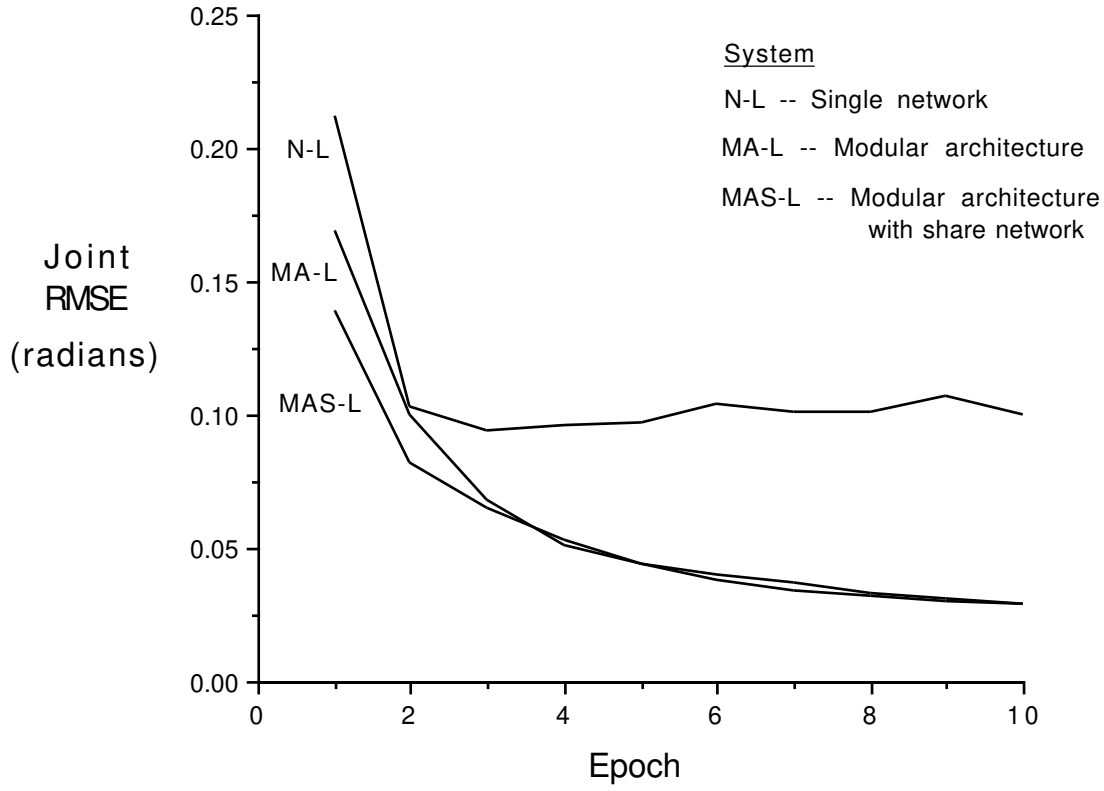

Figure 5.4: Learning curves for the systems trained with the input vectors from set L.

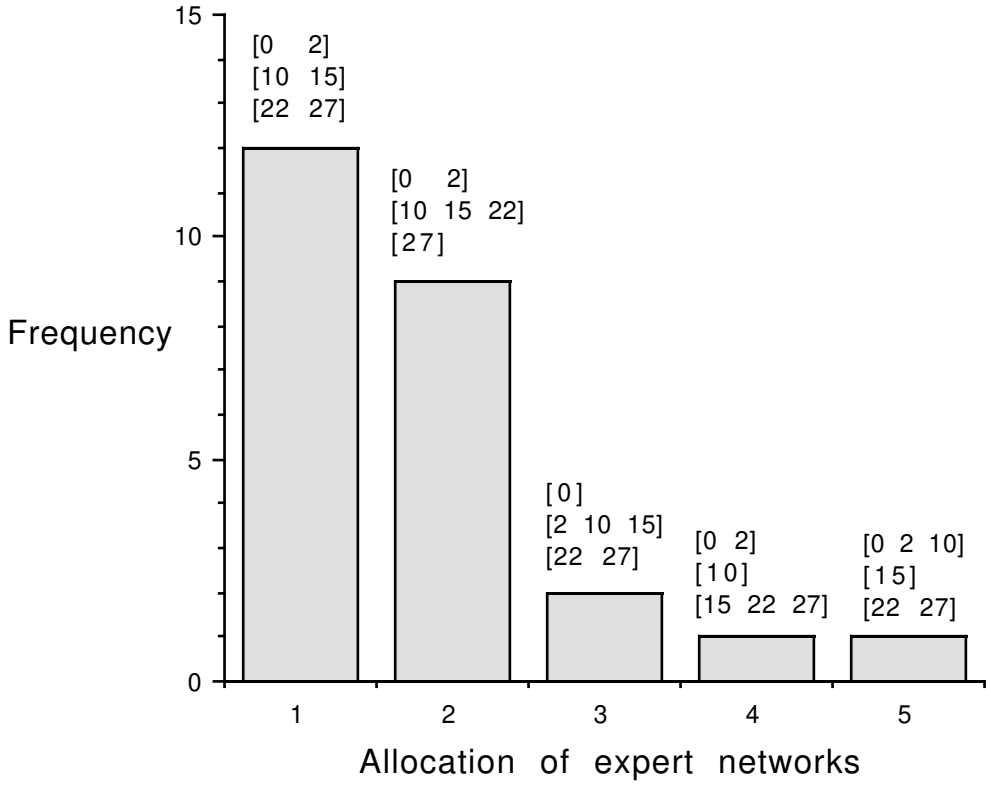

Figure 5.5: Allocation of modular architecture MA–L's expert networks to payloads.

competition when the payload was 10 or 15 kg, and the remaining expert network won the competition when the payload was 22 or 27 kg.

Figure 5.5 shows that the modular architecture MA–L learned to control the robot arm with different payloads by allocating different expert networks to control the arm with payloads from different mass categories. One expert network tended to win the competition to learn to control the arm when there was no payload or a very light payload (0 or 2 kg), another expert network tended to win the competition when

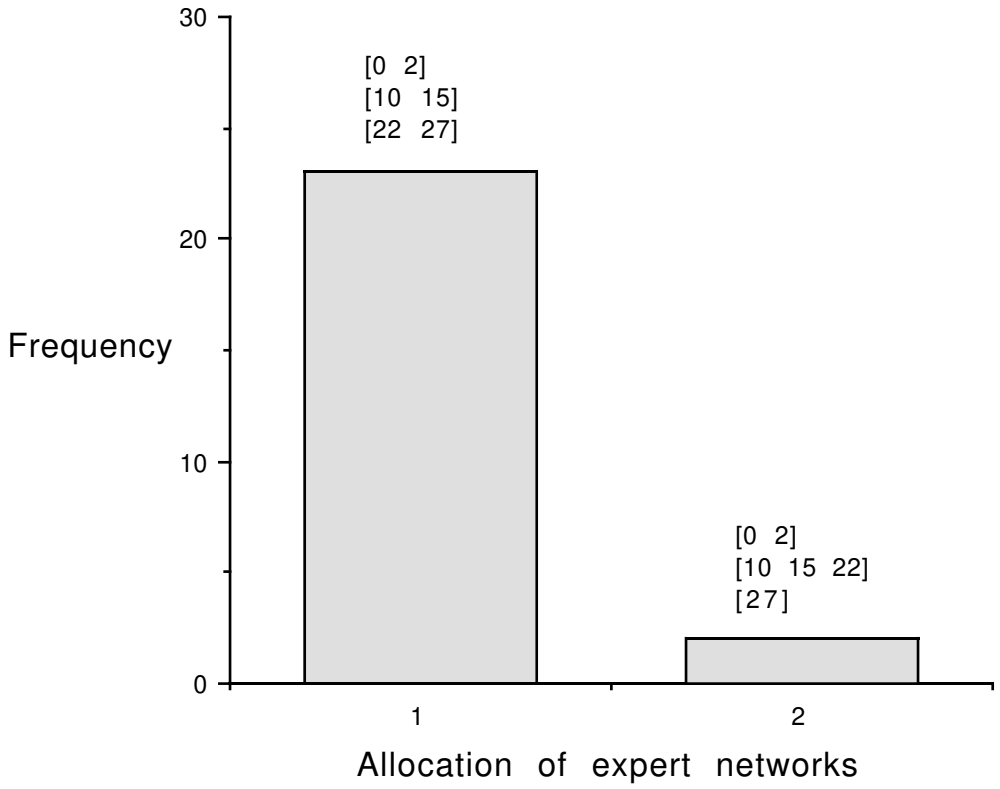

Figure 5.6: Allocation of modular architecture with a share network MAS–L's expert networks to payloads.

the payload was light (10 or 15 kg), and a third expert network tended to win the competition when the payload was heavy (22 or 27 kg). This occurred despite the fact that the system MA–L was only provided with the identity of each payload, not the mass of a payload. The tendency to allocate the same expert network to control the arm with payloads of similar masses, and to allocate different expert networks to control the arm with payloads of dissimilar masses is an outcome of the competitive process. If, for example, an expert network wins the competition to learn to control the arm with no payload, then it is likely to also win the competition when the payload is 2 kg, but lose the competition when the payload is 27 kg. This result suggests that a suitably designed modular architecture tends to allocate the same expert network to similar tasks, resulting in the benefits of positive transfer of training, and different expert networks to dissimilar tasks, thereby avoiding the detrimental effects of negative transfer of training.

As is shown in Figure 5.6, the modular architecture with a share network MAS–L allocated its expert networks to payloads in a manner similar to that of system MA–L. In addition to the torques specified by the share network, one expert network tended to specify extra torques when there was no payload or a very light payload (0 or 2 kg), another expert network tended to specify extra torques when the payload was light (10 or 15 kg), and a third expert network tended to specify extra torques when the payload was heavy (22 or 27 kg). This occurred despite the fact that the system MAS–L was only provided with the identity of each payload, not the mass of a payload. A fact not shown in Figure 5.6 is that the modular architecture with a share network

always discovered the desired task decomposition as defined above. Specifically, this architecture always learned to use the share network and the null expert network to control the robot arm with no payload or a very light payload. The remaining expert networks learned to specify the extra torques required to compensate for the mass of the payload. One expert network specified extra torques when the payload was light, and another expert network specified extra torques when the payload was heavy. This result suggests that a suitably designed modular architecture with a share network tends to learn to perform a task by learning a shared strategy that is used in all contexts along with a set of modifications to this strategy that are applied in a context sensitive manner.

Figure 5.7 shows the learning curves for the systems trained with the input vectors from set NL. The single network N–NL learned faster than the modular architecture MA–NL and the modular architecture with a share network MAS–NL. The relatively slow rates of learning of systems MA–NL and MAS–NL are due to their poor allocation of expert networks to payloads. These allocations are shown in Figures 5.8 and 5.9. Systems MA–NL and MAS–NL did not always allocate different expert networks to control the arm with payloads from different mass categories. These systems often used the control law learned by one of the expert networks to control the arm with payloads of very different masses. Learning is slow because this expert network received conflicting training information when the arm carried payloads of different masses. Because the expert network was not provided with the identity of a payload, the knowledge that it gained about the inverse–dynamics of the arm with one pay-

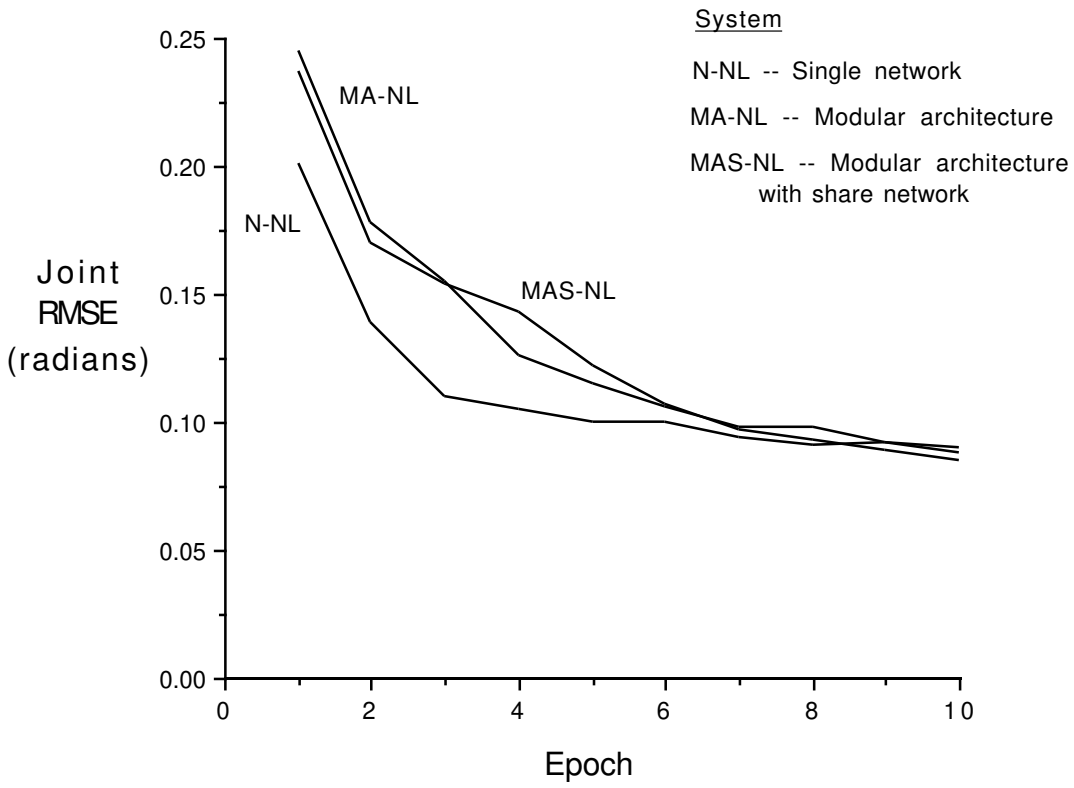

Figure 5.7: Learning curves for the systems trained with the input vectors from set NL.

load was applied without modification to the control of the arm with a payload of a significantly different mass.

Figures 5.5, 5.6, 5.8, and 5.9 show that, in contrast to the modular architectures MA–L and MAS–L, the modular architectures MA–NL and MAS–NL did not allocate their expert networks to the payloads in an appropriate manner. Due to the input vectors they received and the fact that their expert networks do not contain hidden

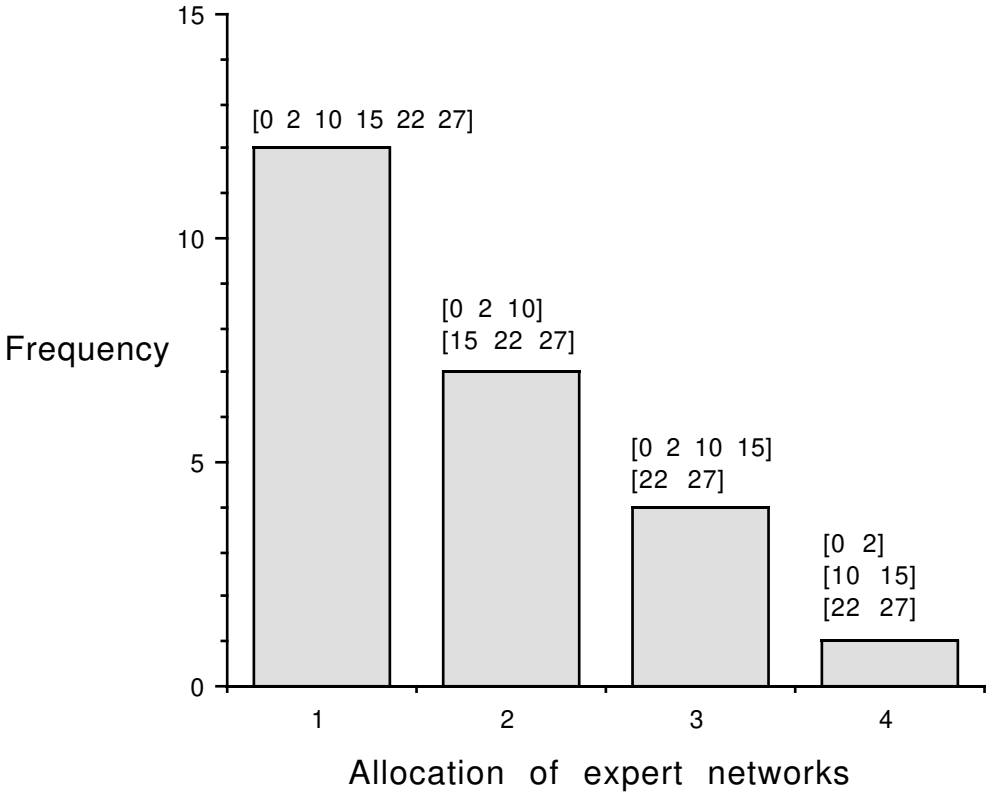

Figure 5.8: Allocation of modular architecture MA–NL's expert networks to payloads.

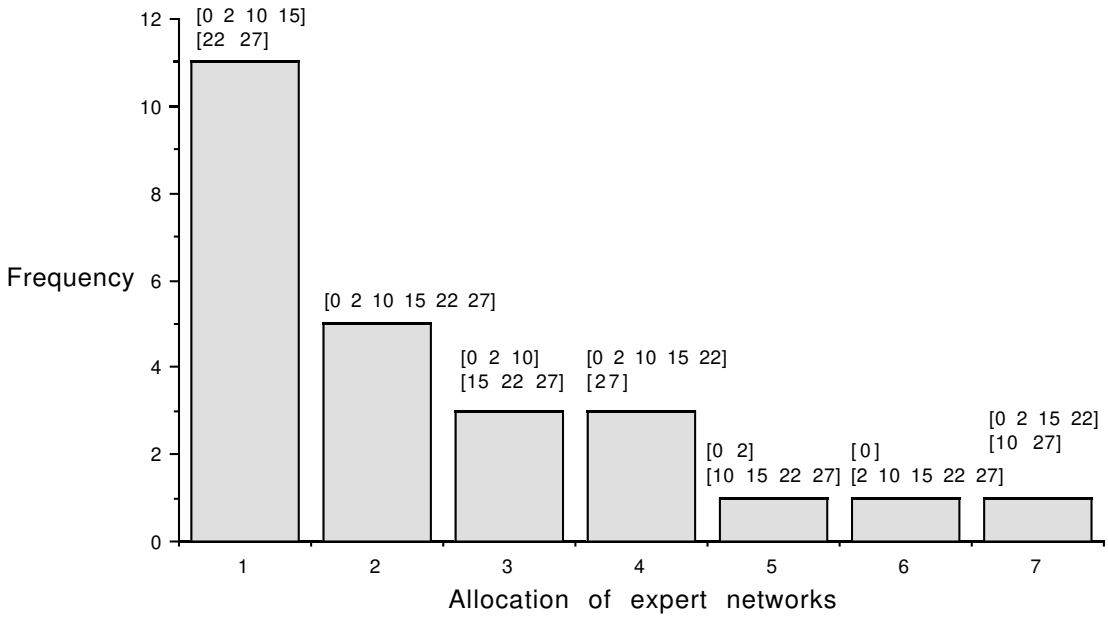

Figure 5.9: Allocation of modular architecture with a share network MAS–NL's expert networks to payloads.

units, the former systems are more constrained in the types of functions that they can compute than the latter systems and, consequently, are more constrained in the types of task decompositions that they can discover. These simulations show that there are many possible decompositions of a task into simpler subtasks. If there are reasons to prefer one decomposition over another, then it is necessary to use domain knowledge in order to design a modular architecture that is appropriately restricted in the types of functions it can compute. The relationship between the design of a modular architecture and the task decomposition that it discovers is an important topic and a complete discussion is postponed until Chapter 6.

5.3 Utilization of Domain Knowledge

In the simulations reported in Section 5.2, the connectionist systems were trained to control the robot arm with various payloads when the systems were only provided with the identity of each payload. In this section, we report simulations in which a system was provided with more detailed information concerning the payload. The system is the modular architecture with a share network MAS–L. Our goal is to demonstrate the ease with which a priori knowledge of the domain may be embedded in modular connectionist architectures.

Three simulations are reported corresponding to three uses of domain knowledge in the training of system MAS–L. In all cases, the modular architecture with a share network learned to allocate its networks in an identical fashion. After training, the

Table 5.4: Three simulations that make use of a priori knowledge of the domain.

Simulation	Stage	Training
1		Assume can detect the absence of a payload.
	1	Train share network using no payload.
	2	Train expert and gating networks as in Section 5.2.
2		Assume can detect the absence of a payload.
		Assume known sample light and heavy payloads.
	1	Train share network using no payload.
	2	Train second expert network using light payload.
	3	Train third expert network using heavy payload.
	4	Train gating network as in Section 5.2.
3		Assume can measure the mass of a payload.
	1	Set weights of gating network.
	2	Train share and expert networks as in Section 5.2.

share network and the null expert network controlled the arm with no payload or with an extremely light payload; the share network and either the second or third expert network controlled the arm with a light payload; and the share network and the remaining expert network controlled the arm with a heavy payload. A description of the three simulations is given below and a summary is provided in Table 5.4. Each simulation is divided into a number of stages. The learning curve for each simulation during the last stage of training is shown in Figure 5.10.

In simulation 1, we assumed that it is possible to detect when the arm has no payload (e.g., the gripper of the arm may be open). The training of the architecture

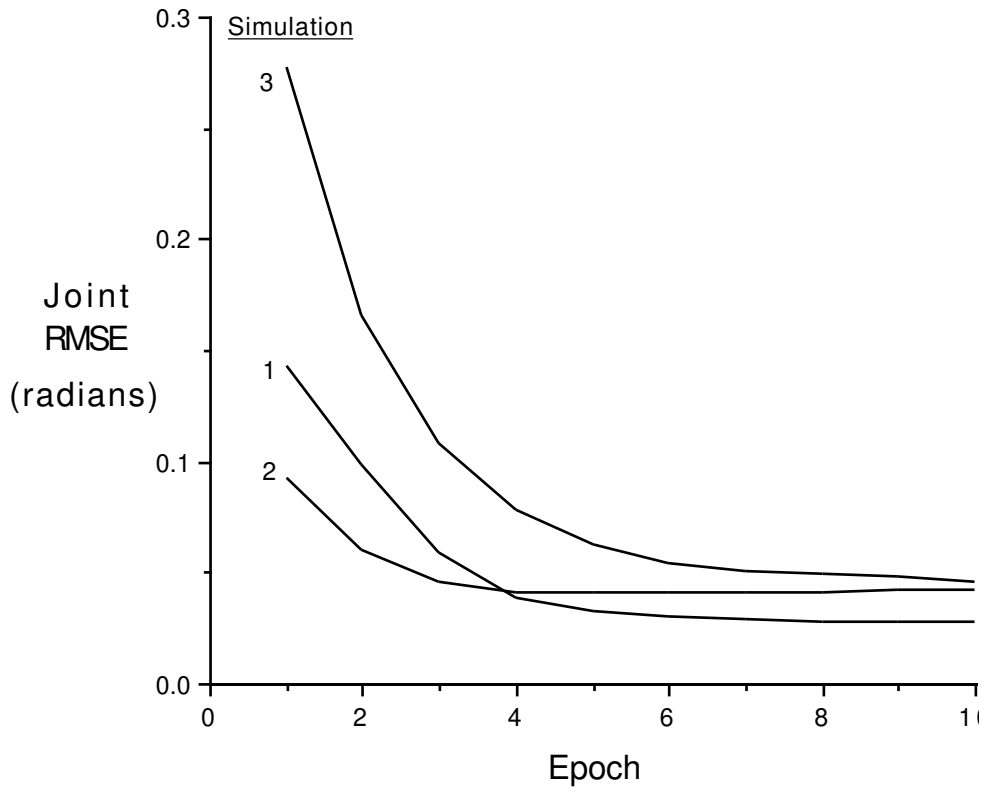

Figure 5.10: Learning curves for the three simulations during the last stage of training.

is divided into two stages. In the first stage, we trained only the share network by requiring it to direct the arm with no payload 50 times along the desired trajectory. After training, the step size of the share network was set to zero. In the second stage, the expert and gating networks were trained as in Section 5.2 to move the arm with a variety of payloads, each of a different mass, along the desired trajectory.

In addition to assuming that it is possible to detect when the arm has no payload, in simulation 2 it was also assumed that we know a sample light payload (12.5 kg) and a sample heavy payload (25 kg). The training of the architecture is divided into four stages corresponding to the training of the share network, the second expert network, the third expert network, and the gating network respectively. At the end of each stage, the step size of the network trained during that stage was set to zero. In the first stage, the share network was trained by requiring it to direct the arm with no payload 50 times along the desired trajectory. In the second stage, the second expert network was trained by requiring it and the share network to direct the arm with the sample light payload 50 times along the desired trajectory. In the third stage, the third expert network was trained by requiring it and the share network to direct the arm with the sample heavy payload 50 times along the desired trajectory. Finally, in the fourth stage, the gating network was trained as in Section 5.2 so that the system could control the arm with a variety of payloads.

In simulation 3, we assumed that the mass of a payload can be measured. The training of the architecture is divided into two stages. In the first stage, the weights of the gating network are hand set so that, in response to the input vector identifying the

payload, the output unit corresponding to the appropriate expert network assumed a value of one and all other output units assumed a value of zero. Specifically, when the robot arm didn't contain a payload or contained a very light payload, g_1 was set to one; when the payload was light, g_2 was set to one; and when the payload was heavy, g_3 was set to one. In the second stage of training, the share and expert networks were trained as in Section 5.2 so that the system could control the arm with a variety of payloads.

These simulations show that if there is sufficient knowledge of a task, some or all of the expert and gating networks can be individually trained independently of the rest of the modular architecture (Hampshire and Waibel [26]). This is one of many methods for embedding domain knowledge in a modular connectionist architecture. Chapter 6 lists other methods for utilizing domain knowledge and argues that a strength of the modular architecture is that its structure is well–suited for incorporating such knowledge.

In summary, this chapter has reported the results of training connectionist systems to control a robot arm to move a variety of payloads, each of a different mass, along a desired trajectory. These results suggest that a suitably designed modular architecture or modular architecture with a share network tends to allocate the same expert network to similar tasks, resulting in the benefits of positive transfer of training, and different expert networks to dissimilar tasks, thereby avoiding the detrimental effects of negative transfer of training. The results also suggest that a suitably designed modular architecture with a share network tends to learn to perform a task by learning a

shared strategy that is used in all contexts along with a set of modifications to this strategy that are applied in a context sensitive manner. Finally, the chapter reported simulations demonstrating that if there is sufficient knowledge of a task, some or all of the expert and gating networks can be individually trained independently of the rest of the modular architecture.