Chapter 6

TASK DECOMPOSITION AND NETWORK ARCHITECTURES

The previous two chapters reported the ability of the modular architecture to perform task decomposition on the "what" and "where" vision tasks and on the multi–payload robotics task. This chapter considers some domain–independent issues concerning task decomposition and discusses similarities between task decomposition and generalization as implemented in connectionist systems.

Because function approximation is an underconstrained problem, networks with too many degrees of freedom may not generalize as desired (e.g., Denker et al. [16], le Cun [42], Poggio and Girosi [58]). One approach to this problem is to use domain knowledge to design a network architecture that is appropriately restricted in the types of functions that it can implement. Such an architecture should generalize in the desired manner. Experience with the modular architecture described here has shown that a similar situation exists with regard to task decomposition. Because function decomposition is an underconstrained problem, there are many possible decompositions of a task into simpler tasks. Thus, modular architectures with too many degrees of freedom may not decompose a task as desired. Section 6.1 reports simulations demonstrating this idea. Section 6.2 argues that 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. Such an architecture should decompose a task in the desired manner.

6.1 Simulations

Three systems were trained to approximate the function

$$f(x) = \begin{cases} \sin x & \text{if } x \in [-2\pi, 0] \\ \sin x + \sin 2x & \text{if } x \in (0, 2\pi]. \end{cases} \tag{6.1}$$

Figure 6.1 shows the graph of this function. At each time step, x was randomly selected from a uniform distribution over the interval $[-2\pi, 2\pi]$. As discussed below, the three systems are modular architectures, and are numbered such that higher–numbered systems are less restricted in the types of functions that they can compute than lower–numbered systems. All systems consist of two expert networks and a gating network (see Table 6.1). The gating network is a single–layer network with 71 input units and 2 output units. The input units provide a coarse–coded representation of x. Details concerning this representation and several other aspects of the simulations are provided in Appendix C.

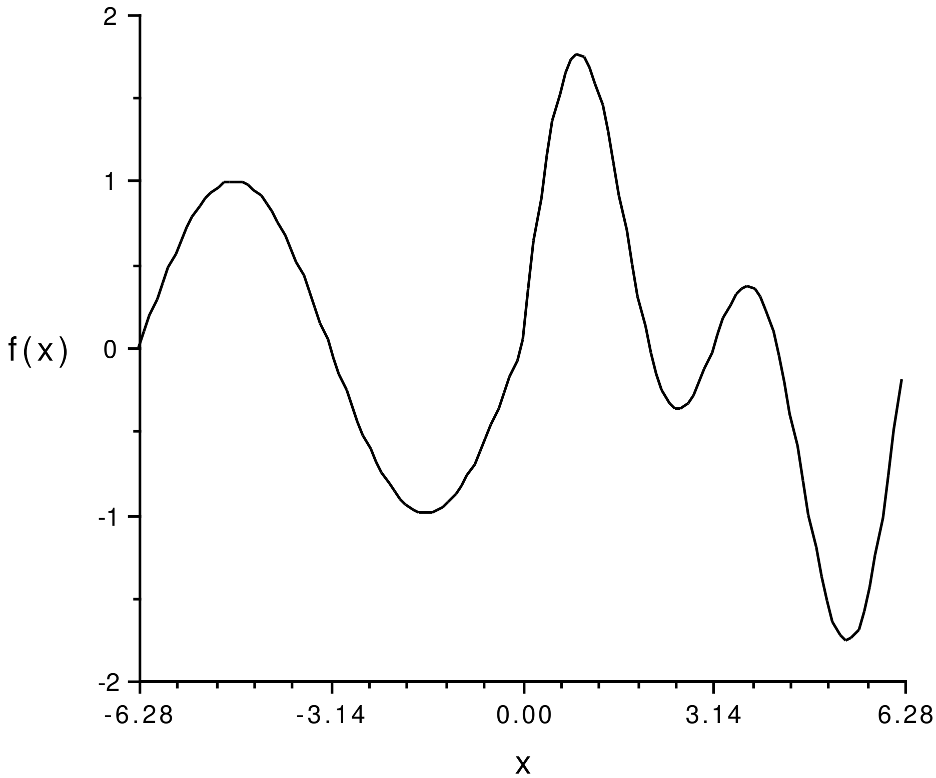

Figure 6.1: Graph of f.

Table 6.1: Systems used in the sine wave experiments.

Modular Architectures				
System	Expert Networks	Expert Networks' Input	Gating Network	Gating Network's Input
1	$2 \rightarrow 1$	$\sin x$ and $\sin 2x$	$71 \rightarrow 2$	coarse–code of x
	$2 \rightarrow 1$	$\sin x$ and $\sin 2x$		
2	$1 \rightarrow 10 \rightarrow 10 \rightarrow 1$	x	$71 \rightarrow 2$	coarse–code of x
	$2 \rightarrow 1$	$\sin x$ and $\sin 2x$		
3	$1 \rightarrow 10 \rightarrow 10 \rightarrow 1$	x	$71 \rightarrow 2$	coarse–code of x
	$1 \rightarrow 10 \rightarrow 10 \rightarrow 1$	x		

In system 1, each expert network is a single–layer network with 2 input units and 1 output unit. The activations of the input units are assigned the values $\sin x$ and $\sin 2x$ respectively. Figure 6.2 shows how system 1 allocated its expert networks at the end of a typical simulation. The horizontal axis gives the values of x. The vertical axis gives the activation values of the output units of the gating network. This graph shows that, roughly, the system allocated one expert network to approximate f when x was negative and the other expert network to approximate f when x was positive. Given the piecewise nature of f, the system discovered an appropriate decomposition. Because of the restrictions placed on the expert networks (they can only compute affine functions of $\sin x$ and $\sin 2x$), this was the only decomposition that the system could use to closely approximate f.

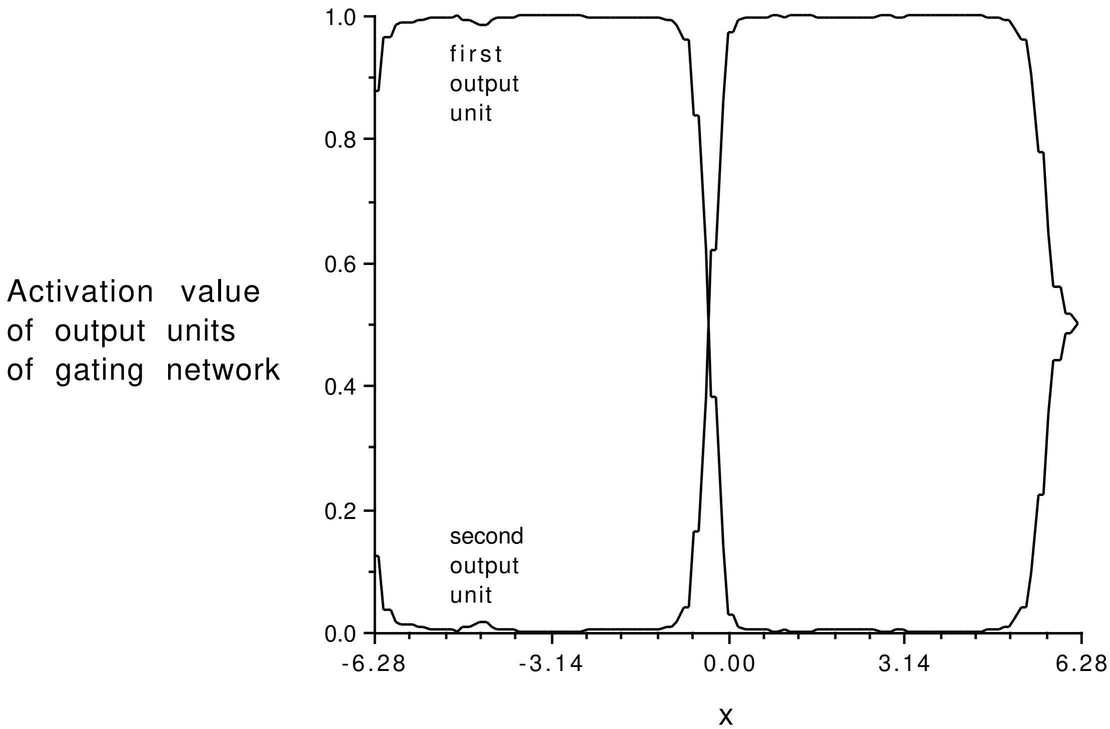

Figure 6.2: Task decomposition discovered by System 1.

Figure 6.2 contains two other noteworthy features. When $x \approx 0$, the outputs of the system were roughly the average of the outputs of the expert networks. This seems appropriate since $x \approx 0$ lies on the borderline that separates the "domains of expertise" of the expert networks. A second feature is that when x was near -2π or 2π, the activations of the output units of the gating network were near 0.5. This is a side–effect of our use of a coarse–coded representation of x and can be eliminated by further training with values of x near these endpoints.

In system 2, the first expert network is a strictly layered network with 1 input unit, 2 layers of 10 hidden units each, and 1 output unit. The input unit encodes the value of x. The second expert network is a single–layer network with 2 input units and 1 output unit. The input units encode the values $\sin x$ and $\sin 2x$. System 2 is less restricted than system 1 in the types of functions it can compute. Consequently, it is more difficult for system 2 to discover a decomposition that allows it to closely approximate f. We find that in order for this system to discover an appropriate decomposition, it is necessary to vary the contribution to the gating network error function (Equation 3.6) of the term responsible for making the outputs of the gating network approach the neutral value, $\frac{1}{n}$, where n is the number of expert networks. This term, the fourth in Equation 3.6, contains the factor λ_{NT} which is non–zero only when the architecture's performance has not significantly improved. Instead of setting λ_{NT} to one when performance has not significantly improved, we initialized it to a value larger than one at the start of training and slowly decreased it to one during training. Recall that this policy of varying λ_{NT} was introduced in

Section 4.1.2, and was found to allow expert networks unable to compete in terms of initial rates of learning—but which may be better in terms of eventual performance—the chance to exert their superiority. When λ_{NT} was modified in this manner, the system consistently discovered an appropriate decomposition.

Figure 6.3 shows how system 2 allocated its expert networks at the end of a typical simulation. This graph shows that the system allocated the multi–layer expert network to approximate f when x was negative and the single–layer expert network to approximate f when x was positive. Although there are an infinite number of task decompositions that would allow the system to closely approximate f, the combination of the following three properties ensured that the system always discovered the decomposition shown in Figure 6.3. These properties are that the single–layer expert network can only compute affine functions of $\sin x$ and $\sin 2x$; the single–layer expert network can learn such affine functions faster than the multi–layer expert network; and it is easier for the multi–layer expert network to learn the function $f(x) = \sin x$ than the function $f(x) = \sin x + \sin 2x$.

In system 3, each expert network is a strictly layered network with 1 input unit, 2 layers of 10 hidden units each, and 1 output unit. The activation of each input unit is assigned the value of x. System 3 is less restricted than either system 1 or 2 in the types of functions that it can compute. As a result, it never discovered the decomposition of f that was discovered by the first two systems. Figure 6.4 shows how system 3 allocated its expert networks at the end of a typical simulation. A comparison of Figures 6.1 and 6.4 shows that, roughly, system 3 allocated one

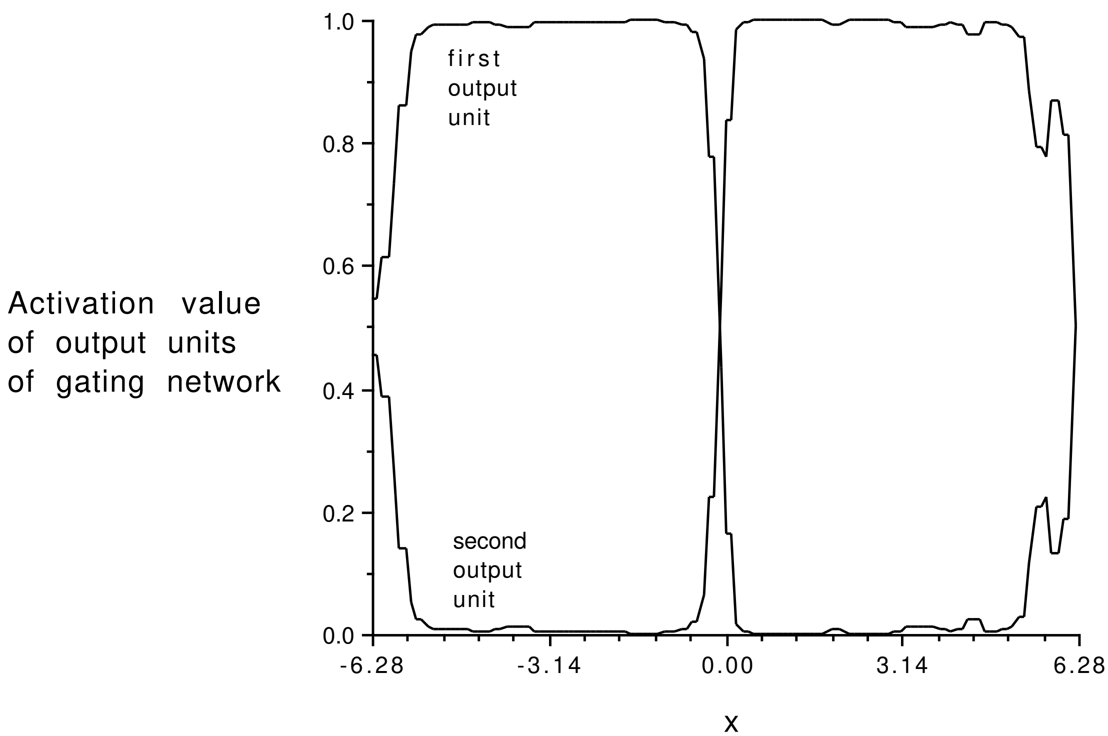

Figure 6.3: Task decomposition discovered by System 2.

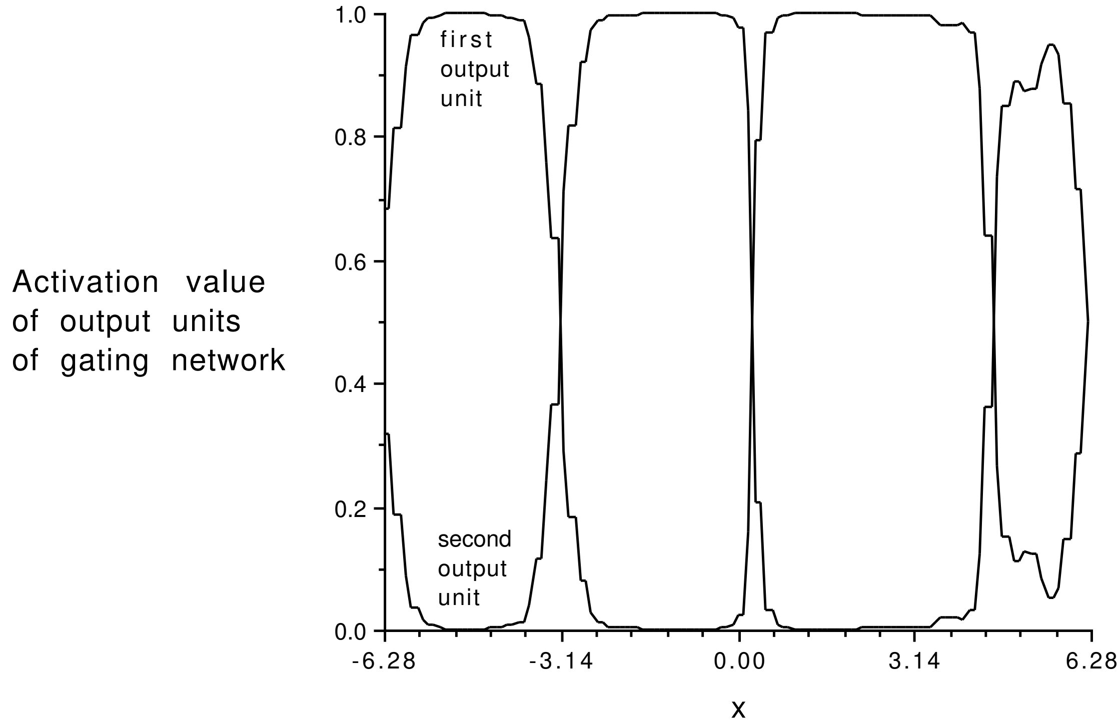

Figure 6.4: Task decomposition discovered by System 3.

expert network to approximate f where f is concave up and allocated the other expert network to approximate f where f is concave down. Although there are an infinite number of task decompositions that would allow the system to closely approximate f, we find that it frequently, though not always, decomposed f in this manner. Considering that there are no constraints on which decomposition the system may discover, this result is somewhat surprising.

As mentioned above, in order to obtain a network that generalizes in a desired manner, it is necessary to design a network architecture that is appropriately restricted in the types of functions it can compute. As the above experiments demonstrate, a similar situation exists with regard to task decomposition. In order to obtain a modular architecture that decomposes a task in a desired manner, it is necessary to design the gating and expert networks so that the architecture is appropriately restricted in the types of functions it can compute.

6.2 Design of Modular Architectures

Designing an appropriate architecture, whether for generalization or task decomposition, requires prior knowledge about the task the system will be required to perform. A strength of modular architectures is that their structures are well–suited for inserting prior knowledge to bias the decompositions to be formed. The experiments reported in this thesis have highlighted several ways in which such knowledge can be utilized. For example, in many tasks there is a natural distinction between information to be processed and information that sets the context for processing. The experimenter may know that when one or more of the (context) inputs remain constant, the mapping from the remaining inputs to the desired outputs is relatively easy to compute. The distinction between the context inputs and the remaining inputs can form the basis for deciding how to divide input information between gating networks and expert networks.

Another way prior knowledge can be incorporated into the design of a modular architecture is that known properties of the function to be approximated can provide constraints on the design of the expert networks. The design of the expert networks, in turn, biases the nature of the decomposition discovered by the modular architecture. This is particularly true when the repertoire of expert networks consists of networks with different characteristics. For example, the function to be approximated may be known to contain a linear portion and a nonlinear portion. In this case, suitable expert networks are easy to design. In general, different expert networks may be designed to possess different topologies, initial weights, activation functions, step sizes, error functions, etc. In addition, different expert networks may receive different input variables or perhaps different representations of the same input variables. Note that we do not advocate providing the modular architecture with a large number of different expert networks. Rather, the experimenter should judiciously design a small set of potentially useful expert networks where the potential utility of an expert network is evaluated using domain knowledge. Indeed, if there is sufficient knowledge of the task, some or all of the expert and gating networks can be individually trained independently of the rest of the architecture (Hampshire and Waibel [26]).

CHAPTER 7

CONCLUSIONS

A novel modular connectionist architecture has been presented in which the networks composing the architecture compete to learn the training patterns. As a result of the competition, different networks learn different training patterns and, thus, learn to compute different functions. The architecture performs task decomposition in the sense that it learns to partition a task into two or more functionally independent tasks and allocates distinct networks to learn each task. In addition, the architecture tends to allocate to each task the network whose topology is most appropriate to that task, and tends to allocate the same network to similar tasks and distinct networks to dissimilar tasks. Furthermore, it 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. These properties were demonstrated by applying the architecture to two sets of tasks. Chapter 4 reported the results of training the architecture to perform object recognition and spatial localization from simulated retinal images. Chapter 5 reported the results of training the architecture to control a simulated robot arm to move a variety of payloads, each of a different mass, along a specified trajectory. Finally, in Chapter 6, it was noted that function decomposition is an underconstrained problem and, thus, different modular architectures may decompose a function in different ways. We argued that a desirable decomposition can be achieved if the architecture is suitably restricted in the types of functions that it can compute. Finding appropriate restrictions is possible through the application of domain knowledge. A strength of the modular architecture is that its structure is well–suited for incorporating domain knowledge.

7.1 Future Work

We conclude this thesis with a list of suggestions for future research using the the modular connectionist architecture. We warn the reader in advance that whereas some of the suggestions are well–formulated, others are highly speculative.

Applications—Although the modular architecture has potential utility in many application areas, we limit our discussion to its use in control tasks. Adaptive control engineers frequently consider control tasks for which it is difficult to design a continuous control law for all regions of a plant's state space. However, these tasks can often be performed by piecewise controllers that employ different control laws when the plant is operating in different regions of its state space (e.g., gain scheduling is a popular piecewise control technique). The success of these controllers in the control of nonlinear systems suggests that task decomposition can be profitably utilized in the performance of these tasks. Control engineers typically design piecewise controllers by first identifying a suitable partition of the state space of the plant, and then designing a useful control law for each region of the state space. One proposed research program would be to go beyond the robotics experiments presented in Chapter 5 by using the modular architecture on more difficult control tasks and comparing the piecewise control laws learned by the architecture with piecewise control laws suggested by adaptive control theory. An advantage of the modular architecture over conventional piecewise controllers is that it can learn both a useful partition of a plant's state space as well as a useful control law for each region of the state space. Therefore, the modular architecture may be applicable to control tasks for which engineers do not possess sufficient a priori knowledge to design an effective piecewise control law.

Connectionist learning theory—A set of studies could investigate the relationship between the design of the modular architecture and the task decomposition discovered by the architecture. These studies would address a number of important issues.

- This thesis has emphasized that the task decomposition discovered by a modular architecture is, in part, determined by the design of the architecture. A second factor determining the task decomposition discovered by the architecture is the order in which training patterns are presented. A proposed research program would attempt to enhance the modular architecture's ability to take advantage of positive transfer of training from earlier to later training patterns. A suitably designed modular architecture may discover task decompositions in which the subtasks identified by the decomposition are also subtasks of other tasks faced by the architecture. This objective emphasizes that the functions learned by the expert networks can be thought of as "building blocks" to be used on other occasions in the performance of more complex tasks.

- The modular architecture both allocates different expert networks to learn different functions as well as combines the expert networks' outputs to determine the output of the system. A proposed research program would decouple these two operations by performing them in different stages of processing. In the first stage, the modular architecture decomposes a task into a set of useful subtasks. A second stage of processing uses a different architecture to combine the expert networks' outputs. For example, many computer vision systems have an early stage of processing during which different modules compute different functions of the visual input. These modules may detect edges or estimate depth, orientation, optic flow, color, etc. In order to interpret the visual image, the modules' outputs are nonlinearly combined during subsequent stages of processing. Given that the modular architecture has similarities with conventional competitive learning systems (see Chapter 3), the use of the modular architecture as a preprocessor may be considered analogous to the use of competitive learning during preprocessing. For example, Moody and Darken [50] trained a network to perform a nonlinear mapping by initially training the hidden units of the network using a competitive learning technique, and subsequently training the output units to combine the outputs of the hidden units using a supervised learning technique.

- In this thesis the expert and gating networks of the modular architecture are trained using different error functions. However, there may be practical and theoretical advantages to training both sets of networks using a single error function. Independently, Geoffrey Hinton (Hinton, personal communication) and Michael Jordan (Jordan, personal communication) developed similar proposals for error functions for training both the expert and gating networks. These error functions are based on the intuition that a competitive learning process analogous to "soft" competitive learning (Bridle [12], Nowlan [54]) may be useful in training all networks of the modular architecture. [1]

- This thesis has described the standard modular connectionist architecture (Figure 3.1) and two modifications of this architecture, namely the modular architecture with multiple gating networks (Figure 4.5) and the modular architecture with a share network (Figure 5.3). Different versions of the architecture may learn different decompositions of the same task. Consequently, different versions are most useful for different tasks. A proposed research program would

[1] The investigation of a single error function for all networks of the modular architecture is currently being pursued in a collaboration between Geoffrey Hinton, Michael Jordan, Steven Nowlan, and the author.

identify other architectural modifications along with the class of tasks for which each modification is most suited.

The notion of modularity has been found to be of considerable utility in cognitive science, particularly in the study of language and vision. Not only have modular theories been found to be more parsimonious and easier to understand than nonmodular theories, but also the predictions of modular theories have in many cases been verified (Freedman and Forster [23]). Modularity is also indispensable in the design and analysis of complex systems in engineering. We feel that the virtues of modular systems cited in the literature of these areas are also relevant to the problem of learning in connectionist networks. In particular, we have argued that if a task can be decomposed into subtasks, each of which has its own idiosyncratic properties, then the learner should itself be a decomposable system in which distinct system resources ("experts") are allocated to distinct subtasks. Such a learning system will in general be more robust, more efficient, and will generalize better than a nonmodular system.

Although domain knowledge may be useful in suggesting an a priori decomposition of a task, the boundaries between subtasks are rarely explicitly marked in the data presented to the learner. Moreover, the optimal allocation of experts to subtasks depends not only on the nature of the task but also on the nature of the learner. For these reasons we have argued that the problem of allocating experts to subtasks is itself part of the learning problem. Even if domain knowledge is used in designing the initial structure of the modular architecture, it is still necessary for the system to discover which experts to assign to which training instances.

The modular architecture presented in this thesis makes use of competition to induce a task decomposition. The competition allows experts to specialize as well as to extend their applicability. We feel that the competition between experts is the essential feature of the approach presented here and should serve as a useful point of departure for the further development of algorithms for learning in modular systems.

APPENDIX A

SIMULATION DETAILS FOR CHAPTER 4

This appendix provides details about the simulations reported in Chapter 4.

Input values—The task bit was set to -1 for the "what" task and to 1 for the "where" task.

Training—The weights of all systems were updated at each time step. Desired output values of 0.1 and 0.9 were used instead of 0 and 1.

Activation functions—The hidden and output units of all systems used activation functions that include the logistic function with asymptotes at 0 and 1.

Initial weights—The weights of the single networks and the expert networks were initialized with values randomly selected from a uniform distribution over the interval $[-\frac{1}{2}, \frac{1}{2}]$. The weights of the gating network were initialized to zero.

Table A.1: Parameter values used in the temporal crosstalk experiments.

	Random training			Blocked training		
System	Step size	Momentum	γ	Step size	Momentum	γ
1	3.00	0.00		0.44	0.75	
2	0.45	0.90		5.25	0.00	
3	5.75	0.00	3.50	6.00	0.00	2.75

Parameter values

- Temporal crosstalk experiments—For the single networks trained with the back-propagation algorithm, we used the step sizes and momentum that roughly give the best performance. These values are listed in Table A.1.

  For some of the parameters of the modular architecture, we searched for the values that give the best behavior, and for others, we didn't. Specifically, we did not attempt to optimize the step size and momentum of the gating network and α used in Equation 3.3. These parameters had the values 0.01, 0.0, and 0.2 respectively. In addition, although each expert network may have its own step size and momentum, the same values were used for all expert networks. Thus, the only parameter values that we attempted to optimize are the step size and momentum used by all expert networks and γ used in Equation 3.4. These values are listed in Table A.1.

- Spatial crosstalk experiment—Since no comparisons were performed in this set of experiments, we did not attempt to locate the optimal parameter values.

Thus, we provide the values that were used, not the values that are best. For the expert networks, the step size and momentum were 8.0 and 0.0. For the gating networks, the step size was 0.01, momentum was 0.0, α was 0.2, and γ was 0.75. When the system's current performance was significantly better than its past performance, λ_{NT} was set to zero. Otherwise, λ_{NT} was 10 for epochs 1–400, 8 for epochs 401–500, 6 for epochs 501–600, 4 for epochs 601–700, 2 for epochs 701–800, and 1 for epochs 801–900.

APPENDIX B

SIMULATION DETAILS FOR CHAPTER 5

This appendix provides details about the simulations reported in Chapter 5.

Training—The weights of all systems were updated at each time step.

Activation functions—The hidden units of all single networks, expert networks, and share networks as well as the output units of the gating networks used activation functions that include the logistic function with asymptotes at 0 and 1. The output units of all single networks, expert networks, and share networks used the identity function in their activation function. If a single network, expert network, or share network contained hidden units, then the output of that network was considered to be 250 times the activation of the output units. The desired outputs for such a network were similarly scaled. This was done so that the activations of the output units lied in the interval [-2, 2] which is close to the interval that contains the activations of the hidden units.

Initial weights—The weights of the single networks, expert networks, and share networks were initialized with values randomly selected from a uniform distribution over the interval $[-\frac{1}{2}, \frac{1}{2}]$. The weights of the gating networks were initialized to zero. The units of all networks except the gating networks contained a bias weight.

Parameter values—For the single networks, we used the step sizes and momentum that roughly give the best performance. For some of the parameters of the modular architectures and the modular architectures with share networks, we searched for the values that give the best behavior, and for others, we didn't. Specifically, we did not attempt to optimize the step size and momentum of the gating networks and α used in Equation 3.3. These parameters had the values 0.01, 0.0, and 0.2 respectively. In addition, although each expert and share network of a system may have its own step size and momentum, the same values were used for all expert and share networks of a system. Thus, the only parameter values that we attempted to optimize are the step size and momentum used by all expert and share networks of a system and γ used in Equation 3.4. The parameter values used are listed in Table B.1.

Table B.1: Parameter values used in the multi–payload robotics experiments.

System	Step size	Momentum	γ
N–L	0.00400	0.00	
N–NL	0.00225	0.25	
MA–L	0.00100	0.00	1.2
MA–NL	0.00165	0.25	2.2
MAS–L	0.00070	0.00	1.3
MAS–NL	0.00075	0.25	1.3

APPENDIX C

SIMULATION DETAILS FOR CHAPTER 6

This appendix provides details about the simulations reported in Chapter 6.

Input values—The input units of the gating network provided a coarse-coded representation of the value of x. The n^{th} input unit was set to 1 if

$$-2\pi + \frac{(n-1)\pi}{20} \leq x < -2\pi + \frac{(n-1)\pi}{20} + \frac{\pi}{2}. \tag{C.1}$$

Otherwise, it was set to 0.

Training—The weights of all systems were updated at each time step. The first system was run for 50,000 time steps. The second and third systems were run for 100,000 time steps.

Activation functions—The hidden units of all expert networks and the output units of the gating network used activation functions that include the logistic function with asymptotes at 0 and 1. The output units of all expert networks used the identity function in their activation function.

Initial weights—The weights of the expert networks were initialized with values randomly selected from a uniform distribution over the interval $[-\frac{1}{2}, \frac{1}{2}]$. The weights of the gating network were initialized to zero. The units of the expert networks, but not the gating network, contained a bias weight.

Parameter values

- Gating network—In all three systems, the values of the parameters of the gating network were: step size = 0.01; momentum = 0.0; $\alpha = 0.2$ (used in Equation 3.3); $\gamma = 0.6$ (used in Equation 3.4).

- System 1—Both expert networks used a step size of 0.005 and momentum of 0.0.

- System 2—The step size of the multi–layer expert network was 0.1 and the step size of the single–layer expert network was 0.005. Both expert networks used a momentum of 0.0. When the system's current performance was significantly better than its past performance, λ_{NT} was set to zero. Otherwise, λ_{NT} was 10 for time steps 1–15000, 8 for time steps 15001–30000, 6 for time steps 30001–45000, 4 for time steps 45001–60000, 2 for time steps 60001–75000, and 1 for time steps 75001-100000.

- System 3—Both expert networks used a step size of 0.1 and momentum of 0.0.

REFERENCES

[1] Albus, J.S. (1975) A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamical Systems: Measurement and Control*, 97, 220–227.

[2] Atkeson, C.G. & McIntyre, J. (1986) Robot trajectory learning through practice. *Proceedings of the IEEE Conference on Robotics and Automation*, 1737–1742.

[3] Atkeson, C.G. & Reinkensmeyer, D.J. (1988) Using associative content–addressable memories to control robots. *Proceedings of the IEEE Conference on Decision and Control*, 792–797.

[4] Ballard, D.H. (1984) Parameter Nets. *Artificial Intelligence*, 22, 235–267.

[5] Ballard, D.H. (1986) Cortical connections and parallel processing: Structure and function. *The Behavioral and Brain Sciences*, 9, 67–120.

[6] Ballard, D.H. (1987) Modular learning in neural networks. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 279–284.

[7] Barlow, H.B. (1986) Why have multiple cortical areas? *Vision Research*, 26, 81–90.

[8] Barlow, H. & Földiák, P. (1989) Adaptation and decorrelation in the cortex. In R. Durbin, C. Miall, & G. Mitchison (eds.), *The Computing Neuron.* Reading, MA: Addison–Wesley Publishing Company.

[9] Barto, A.G. & Anandan, P. (1985) Pattern–recognizing stochastic learning automata. *IEEE Transactions On Systems, Man, and Cybernetics*, 15, 360–375.

[10] Barto, A.G. & Jordan, M.I. (1987) Gradient following without back–propagation in layered networks. *Proceedings of the IEEE First Annual Conference on Neural Networks*, 2, 629–636.

[11] Barto, A.G., Sutton, R.S., & Watkins, C.J.C.H. (1989) Learning and sequential decision making. COINS Technical Report 89–95, University of Massachusetts, Amherst, MA. A version of this report will appear in M. Gabriel & J.W. Moore (eds.), *Learning and Computational Neuroscience.* Cambridge, MA: The MIT Press, in press.

[12] Bridle, J. (1989) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fougelman–Soulie & J. Herault (eds.), *Neuro–computing: Algorithms, Architectures, and Applications.* New York: Springer–Verlag.

[13] Cowey, A. (1981) Why are there so many visual areas? In F.O. Schmidt, F.G. Warden, G. Adelman, & S.G. Dennis (eds.), *The Organization of the Cerebral Cortex.* Cambridge, MA: The MIT Press.

[14] Cowey, A. (1985) Aspects of cortical organization related to selective attention and selective impairments of visual perception: A tutorial review. In M.I. Posner & O.S.M. Marin (eds.), *Attention and Performance XI.* Hillsdale, NJ: Lawrence Erlbaum Associates.

[15] Craig, J.J. (1986) *Introduction to Robotics: Mechanics and Control.* Reading, MA: Addison–Wesley Publishing Company.

[16] Denker, J., Schwartz, D., Wittner, B., Solla, S., Hopfield, J., Howard, R., & Jackel, L. (1987) Automatic learning, rule extraction, and generalization. *Complex Systems*, 1, 877–922.

[17] Duda, R.O. & Hart, P.E. (1973) *Pattern Classification and Scene Analysis.* New York: John Wiley & Sons.

[18] Durbin, R. & Mitchison, G. (1990) A dimension reduction framework for understanding cortical maps. *Nature*, 343, 644–647.

[19] Durbin, R. & Willshaw, D.J. (1987) An analogue approach to the traveling salesman problem using an elastic net method. *Nature*, 326, 689–691.

[20] Feldman, J.A. (1982) Dynamic connections in neural networks. *Biological Cybernetics*, 46, 27–39.

[21] Feldman, J.A. & Ballard, D.H. (1982) Connectionist models and their properties. *Cognitive Science*, 6, 205–254.

[22] Fodor, J.A. (1983) *The Modularity of Mind.* Cambridge, MA: The MIT Press.

[23] Freedman, S.A. & Forster, K.I. (1985) The psychological status of overgenerated sentences. *Cognition*, 19, 101–131.

[24] Geschwind, N. & Galaburda, A.M. (1987) *Cerebral Lateralization: Biological Mechanisms, Associations, and Pathology.* Cambridge, MA: The MIT Press.

[25] Grossberg, S. (1987) Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11, 23–63.

[26] Hampshire, J.B. & Waibel, A.H. (1989) The meta–pi network: Building distributed knowledge representations for robust pattern recognition. Technical Report CMU–CS–89–166, Carnegie–Mellon University, Pittsburgh, PA.

[27] Hinton, G.E. (1981) A parallel computation that assigns canonical object–based frames of reference. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 683–685.

[28] Hinton, G.E. (1981) Shape representation in parallel systems. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1088–1096.

[29] Hinton, G.E. (1989) Connectionist learning procedures. *Artificial Intelligence*, 40, 185–234.

[30] Hornik, K., Stinchcombe, M., & White, H. (1989) Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359–366.

[31] Jacobs, R.A. (1989) Initial experiments on constructing domains of experience and hierarchies in connectionist systems. In D. Touretzky, G. Hinton, & T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann Publishers.

[32] Jacobs, R.A., Jordan, M.I., & Barto, A.G. (1990) Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. COINS Technical Report 90–27, University of Massachusetts, Amherst, MA.

[33] Jordan, M.I. (1986) Attractor dynamics and parallelism in a connectionist sequential machine. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 531–546.

[34] Jordan, M.I. (1986) Serial order: A parallel, distributed processing approach. Technical Report ICS–8604, University of California at San Diego, La Jolla, CA. A version of this report will appear in J.L. Elman & D.E. Rumelhart (eds.), *Advances in Connectionist Theory: Speech.* Hillsdale, NJ: Lawrence Erlbaum Associates, in press.

[35] Jordan, M.I. (1988) Supervised learning and systems with excess degrees of freedom. COINS Technical Report 88–27, University of Massachusetts, Amherst, MA.

[36] Kaas, J.H. (1982) The segregation of function in the nervous system: Why do sensory systems have so many subdivisions? In W.P. Neff (ed.), *Contributions to Sensory Physiology (Volume 7).* New York: Academic Press.

[37] Kawato, M. (in press) Computational schemes and neural network models for formation and control of multijoint arm trajectory. In T. Miller, R.S. Sutton, & P.J. Werbos (eds.), *Neural Networks for Control.* Cambridge, MA: The MIT Press.

[38] Kawato, M., Furukawa, K., & Suzuki, R. (1987) Hierarchical neural–network model for control and learning of voluntary movement. *Biological Cybernetics*, 57, 169–185.

[39] Kohonen, T. (1982) Self–organized formation of topologically correct feature maps. *Biological Cybernetics*, 43, 56–69.

[40] Kosslyn, S.M. (1987) Seeing and imagining in the cerebral hemispheres: A computational approach. *Psychological Review*, 94, 148–175.

[41] le Cun, Y. (1985) Une procedure d'apprentissage pour reseau a sequil assymetrique [A learning procedure for asymmetric threshold network]. *Proceedings of Cognitiva*, 85, 599–604.

[42] le Cun, Y. (1989) Generalization and network design strategies. Technical Report CRG–TR–89–4, University of Toronto, Toronto, Ontario.

[43] Marr, D. (1982) *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* New York: W.H. Freeman and Company.

[44] Maxwell, T., Giles, C.L., Lee, Y.C., & Chen, H.H. (1986) Nonlinear dynamics of artificial neural systems. In J.S. Denker (ed.), *Neural Networks for Computing, AIP Conference Proceedings 151.* New York: American Institute of Physics.

[45] McClelland, J.L. (1986) The programmable blackboard model of reading. In J.L. McClelland, D.E. Rumelhart, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models.* Cambridge, MA: The MIT Press.

[46] Miller, W.T. (1987) Sensor based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation*, 3, 157–165.

[47] Miller, W.T., Glanz, F.H., & Kraft, L.G. (1987) Application of a general learning algorithm to the control of robotic manipulators. *The International Journal of Robotics Research*, 6, 84–98.

[48] Minsky, M. (1986) *The Society of Mind.* New York: Simon and Schuster.

[49] Mishkin, M., Ungerleider, L.G., & Macko, K.A. (1983) Object vision and spatial vision: Two cortical pathways. *Trends In Neurosciences*, 6, 414–417.

[50] Moody, J. & Darken, C.J. (1989) Fast learning in networks of locally–tuned processing units. *Neural Computation*, 1, 281–294.

[51] Narendra, K. & Thathachar, M.A.L. (1989) *Learning Automata: An Introduction.* Englewood Cliffs, NJ: Prentice Hall.

[52] Nowlan, S.J. (1989) The hard vs soft distinction in competitive adaptation. Thesis Proposal, Computer Science Department, Carnegie Mellon University.

[53] Nowlan, S.J. (1990) Max likelihood competition in RBF networks. Technical Report CRG–TR–90–2, Connectionist Research Group, University of Toronto, Toronto, Ontario.

[54] Nowlan, S.J. (1990) Maximum likelihood competitive learning. In D.S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2.* San Mateo, CA: Morgan Kaufmann Publishers.

[55] Nowlan, S.J. & Hinton, G.E. (1989) Maximum likelihood decision–directed adaptive equalization. Technical Report CRG–TR–89–8, Connectionist Research Group, University of Toronto, Toronto, Ontario.

[56] Parker, D.B. (1985) Learning logic. Technical Report TR–47, Massachusetts Institute of Technology, Cambridge, MA.

[57] Plaut, D.C. & Hinton, G.E. (1987) Learning sets of filters using back–propagation. *Computer Speech and Language*, 2, 35–61.

[58] Poggio, T. & Girosi, F. (1989) A theory of networks for approximation and learning. A.I. Memo No. 1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

[59] Pollack, J.B. (1987) Cascaded back–propagation on dynamic connectionist networks. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, 391–404.

[60] Pomerleau, D.A. (1987) The meta–generalized delta rule: A new algorithm for learning in connectionist networks. Technical Report CMU–CS–87–165, Carnegie–Mellon University, Pittsburgh, PA.

[61] Reggia, J.A. (1987) Properties of a competition–based activation mechanism in neuromimetic network models. *IEEE First International Conference on Neural Networks*, 2, 131–138.

[62] Rueckl, J.G., Cave, K.R., & Kosslyn, S.M. (1989) Why are "what" and "where" processed by separate cortical visual systems? A computational investigation. *Journal of Cognitive Neuroscience*, 1, 171–186.

[63] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986) Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: The MIT Press.

[64] Rumelhart, D.E. & Zipser, D. (1986) Feature discovery by competitive learning. In D.E. Rumelhart, J.L. McClelland, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: The MIT Press.

[65] Sejnowski, T.J. (1981) Skeleton filters in the brain. In G.E. Hinton & J.A. Anderson (eds.), *Parallel Models of Associative Memory.* Hillsdale, NJ: Lawrence Erlbaum Associates.

[66] Selfridge, O., Sutton, R.S., & Barto, A.G. (1985) Training and tracking in robotics. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 670–672.

[67] Sutton, R.S. (1986) Two problems with backpropagation and other steepest–descent learning procedures for networks. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 823–831.

[68] Sutton, R.S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 9–44.

[69] Waibel, A. (1989) Modular construction of time–delay neural networks for speech recognition. *Neural Computation*, 1, 39–46.

[70] Werbos, P.J. (1974) *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA.

[71] Werbos, P.J. (1988) Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions of Systems, Man, and Cybernetics*, 17, 7–20.

[72] Yeung, D.Y. & Bekey, G.A. (1989) Using a context–sensitive learning network for robot arm control. *Proceedings of the 1989 IEEE Conference on Robotics and Automation*, 3, 1441–1447.

[73] Yuille, A.L. & Grzywacz, N.M. (1989) A winner–take–all mechanism based on presynaptic inhibition feedback. *Neural Computation*, 1, 334–347.