CHAPTER 3

A MODULAR CONNECTIONIST ARCHITECTURE

In this chapter we introduce a modular connectionist architecture that learns to partition a task into two or more functionally independent tasks and allocates distinct networks to learn each task.

3.1 Output of the Architecture

The architecture illustrated in Figure 3.1 consists of two types of networks: *expert networks* and a *gating network*. The expert networks compete to learn the training patterns, and the gating network mediates this competition. After training, expert networks 1 and 2 compute different functions that are useful in different regions of the input space. Let the vectors $\mathbf{y}_1$ and $\mathbf{y}_2$ denote the outputs of the two expert networks. The gating network is an administrative agency that decides whether expert network 1 or 2 is currently applicable. Let the scalars g_1 and g_2 denote the two output units of

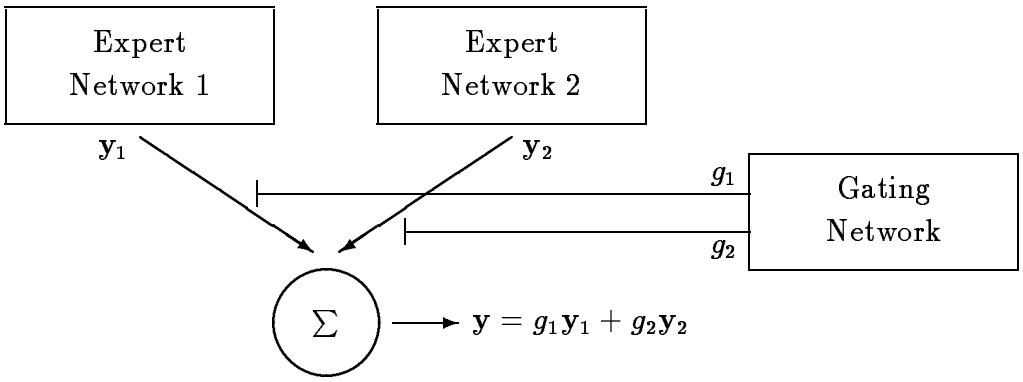

Figure 3.1: A modular connectionist architecture.

the gating network. The output of the entire architecture, $\mathbf{y}$, is $g_1\mathbf{y}_1 + g_2\mathbf{y}_2$. Therefore, when $g_1 = 1$ and $g_2 = 0$, expert network 1 determines the output of the architecture, whereas when $g_1 = 0$ and $g_2 = 1$, expert network 2 determines the output. In general, the architecture may contain any number of expert networks. If there are n expert networks, then the gating network has n output units, and the architecture's output is

$$\mathbf{y} = \sum_{i=1}^{n} g_i \mathbf{y}_i. \tag{3.1}$$

3.2 Training of the Architecture

During training, the weights of all networks are modified simultaneously using the backpropagation algorithm. However, the learning rules used to train the expert and gating networks are based on the minimization of different error functions. At each

time step, the weights of the expert networks are modified so as to reduce the sum of squared error between the output of the system, $\mathbf{y}$, and the desired output, $\mathbf{y}^*$. This error, denoted $J_{\mathbf{y}}$, is given by

$$J_{\mathbf{y}} = \frac{1}{2}(\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y}). \tag{3.2}$$

The weights of the gating network are modified so as to reduce a more complicated error function. The intuition behind this function is as follows. For each training pattern, one expert network comes closer to producing the desired output than the other expert networks. In the competition among networks, this one is called the 'winner' and all others are called 'losers'. If on a given training pattern, the system's performance is significantly better than it has been in the past, then the weights of the gating network are adjusted to make the output corresponding to the winning expert network increase towards one and the outputs corresponding to the losing expert networks decrease towards zero. Alternatively, if the system's performance has not improved, then the gating network's weights are adjusted to move all of its outputs towards some neutral value.

This intuition is expressed mathematically as follows. First, it is necessary to specify what it means for the system's performance to be significantly better than it was in the past. If t is the current time step, then the error $J_{\mathbf{y}}(t)$ is a measure of the current performance. We measure the system's past performance by forming

an exponentially weighted average of $J_{\mathbf{y}}$ over time steps earlier than t. This value, denoted $\overline{J_{\mathbf{y}}}$, is computed iteratively by the following difference equation:

$$\overline{J_{\mathbf{y}}}(t) = \alpha J_{\mathbf{y}}(t) + (1-\alpha)\overline{J_{\mathbf{y}}}(t-1), \tag{3.3}$$

where α, $0 \leq \alpha < 1$, determines how rapidly past values of $J_{\mathbf{y}}$ are forgotten. We use binary variables λ_{WTA} (WTA stands for 'winner–take–all') and λ_{NT} (NT stands for 'neutral') to indicate whether the system's performance has significantly improved. Specifically,

$$\begin{aligned} \text{If} \qquad & J_{\mathbf{y}}(t) < \gamma\overline{J_{\mathbf{y}}}(t-1), \\ & \text{Then } \lambda_{WTA} = 1 \text{ and } \lambda_{NT} = 0 \\ & \text{Else } \lambda_{WTA} = 0 \text{ and } \lambda_{NT} = 1, \end{aligned} \tag{3.4}$$

where γ is a multiplicative factor that determines how much less the current error must be than the measure of past errors in order for the system's performance to be considered significantly improved.

If the architecture's performance has significantly improved ($\lambda_{WTA} = 1$), we determine which expert network's output is closest to the desired output. Define the error for expert network i to be the sum of squared error between the expert network's output, $\mathbf{y}_i$, and the desired output, $\mathbf{y}^*$. This value, denoted $J_{\mathbf{y}_i}$, is

$$J_{\mathbf{y}_i} = \frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_i). \tag{3.5}$$

The winning expert network is the network with the smallest error. If expert network i is the winner, then the desired value of the i^{th} output unit of the gating network, denoted g_i^*, is set to one. Otherwise, if expert network i is a loser, g_i^* is set to zero. If the architecture's performance has not significantly improved ($\lambda_{NT} = 1$), then the weights of the gating network are adjusted so that all the outputs of the gating network are moved towards a neutral value. This value is $\frac{1}{n}$, where n is the number of expert networks.

Using the quantities defined above, it is possible to write the gating network's error function, J_G, as:

$$\begin{aligned} J_G &= \lambda_{WTA}\frac{1}{2}\sum_{i=1}^{n}(g_i^* - g_i)^2 + \\ &\quad \lambda_{WTA}\frac{1}{2}(1 - \sum_{i=1}^{n} g_i)^2 + \\ &\quad \lambda_{WTA}\sum_{i=1}^{n} g_i(1 - g_i) + \\ &\quad \lambda_{NT}\frac{1}{2}\sum_{i=1}^{n}(\frac{1}{n} - g_i)^2. \end{aligned} \tag{3.6}$$

Due to the definition of λ_{WTA} and λ_{NT} (Equation 3.4), the first three terms of Equation 3.6 contribute to the error when the architecture's performance has significantly improved, whereas only the fourth term contributes to the error when the system's performance has not significantly improved. The first term is the sum of squared error between the desired outputs and the actual outputs of the gating network. The

second term takes its smallest value when the outputs of the gating network sum to one. The third term takes its smallest value when the outputs of the gating network are binary valued. Therefore, the effect of changing the gating network's weights to reduce the second and third terms is that, in response to each input pattern, one output unit approaches one and all others approach zero. The fourth term is the sum of squared error between the neutral value and the actual outputs of the gating network. Reducing this term, which occurs only when the architecture's performance has not significantly improved, causes all of the outputs of the gating network to approach the neutral value $\frac{1}{n}$.

3.3 Discussion

The equations given above imply that there are three types of interactions among the networks of the modular architecture. The first type of interaction is that the gating network determines how much each expert network contributes to the output of the system (Equation 3.1); the second is that the performances of the expert networks determine the desired outputs of the gating network; and the third is that the gating network determines how much each expert network learns about each training pattern.

Referring to Figure 3.1, the error vector backpropagated into expert network 1 is $g_1(\mathbf{y}^* - \mathbf{y})$, and the error vector backpropagated into expert network 2 is $g_2(\mathbf{y}^* - \mathbf{y})$. This means that, in addition to determining how much each expert network contributes to the output of the architecture, the gating network also determines the

magnitudes of the expert networks' error vectors, and therefore determines how much each expert network learns about each training pattern. This interaction between the expert networks and the gating network implements a kind of credit assignment policy whose ramifications can be clarified by the following two examples.

Suppose that the gating network responds to an input pattern with $g_1 = 1$ and $g_2 = 0$. This implies that the output of the architecture, $\mathbf{y}$, is the output of expert network 1. In this case, the error vector backpropagated to expert network 1 is $\mathbf{y}^* - \mathbf{y}$, and the error vector backpropagated to expert network 2 is the zero vector. Thus, the first expert network is the one that learns about the function that generated the current training pattern, and the second expert network does not adjust its weights at all. This assignment of credit is logical because expert network 1 is solely responsible for the output of the architecture.

As a second example, suppose that during training of the architecture, expert network 1 is more closely approximating a given training pattern than is expert network 2, and g_1 is slightly larger than g_2. In this case, the first expert network receives a larger error and learns more about the function that generated the training pattern than the second expert network. Consequently, expert network 1 learns to perform this function even better than expert network 2, which causes g_1 to grow even larger than g_2. Thus, this credit assignment policy produces a positive feedback effect in the sense that it enhances the performance advantage of the expert network that is already most closely approximating the current target vector.

This credit assignment policy causes the modular architecture to allocate different expert networks to different tasks. Due to the positive feedback effect, one expert network learns the training patterns that compose a task. However, when later presented with the patterns that compose a second task, the network that won the competition to learn the patterns from the first task is unlikely to also win the competition to learn the new training patterns (unless the two tasks are very similar). Therefore, a different expert network wins the competition to learn the new training patterns. A consequence of the competition to learn the training patterns is that different expert networks learn to perform different tasks.

3.4 Relationship to Previous Research

Competitive Learning—There are many similarities between the competitive learning performed by conventional competitive learning systems (e.g., Kohonen [39], Rumelhart and Zipser [64], Grossberg [25], Reggia [61], Durbin and Willshaw [19], Yuille and Grzywacz [73]) and that performed by the modular architecture. Both types of systems utilize a competition among its components. In conventional competitive learning systems, the competition is between the units of a single network. In the modular architecture, the competition is between the different expert networks. Both types of systems have a method for selecting the winner of the competition. In conventional competitive learning systems, the competition is unsupervised. The winner of the competition at each time step is the unit whose weight vector most

closely matches the input vector. In the modular architecture, the competition is based on supervised errors. The winner of the competition at each time step is the expert network whose output vector most closely matches the desired output vector (Equation 3.5). Both types of systems utilize a competition to learn to respond to the input patterns. In conventional competitive learning systems, the units of a network compete for the right to respond maximally to each input pattern. In the modular architecture, the expert networks compete for the right to learn to produce the desired output pattern for each input pattern. Both types of systems attempt to discover natural groupings of patterns. A goal of conventional competitive learning systems is to cluster the input patterns into natural groupings. Different units learn to respond maximally to input patterns from different groupings. A goal of the modular architecture is to cluster the training patterns, which are input patterns together with desired output patterns, into natural groupings. Different expert networks learn training patterns from different groupings.

Despite the similarities between conventional competitive learning systems and the modular architecture, analogies between these two types of systems are imperfect. Such analogies fail for both implementational and theoretical reasons. At the implementational level, the training of the expert networks is closer to "soft" competitive learning than conventional winner–take–all or "hard" competitive learning. [1]

[1]Whereas "hard" competitive learning processes modify the parameters of only the winning competitor at each time step, "soft" competitive learning processes modify the parameters of all competitors in proportion to how well each competitor did in the competition for the current input

As discussed above, all expert networks, not just the winning expert network, modify their weights at each time step. The magnitude of an expert network's error vector is proportional to the output of the gating network corresponding to that expert network. This output is, in turn, related to how the expert network has fared in the competition for the current training pattern. Consequently, each expert network modifies its weights in proportion to how well it has fared in the competition for the current training pattern. The training of the gating network is also not done in a conventional winner–take–all manner. As explained above, the modification of the weights of the gating network is based on the selection of the winning expert network as well as on whether the architecture's performance has significantly improved.

At the theoretical level, analogies between conventional competitive learning systems and the modular architecture are also inexact. The clustering of input patterns discovered by many competitive learning systems can be characterized as having minimal within–group variance and maximal between–group variance. Such a characterization is possible because of the existence of metrics that allow one to compute the "nearness" or "similarity" of input patterns. In contrast, the clustering of training patterns discovered by the modular architecture cannot be similarly characterized. Such a characterization is not possible because of the difficulty of defining a metric that allows us to compute the "nearness" or "similarity" of training patterns. The problem of defining a metric for training patterns has been discussed in the con-

pattern (see Bridle [12], Nowlan [52, 53, 54], and Nowlan and Hinton [55] for a more thorough discussion of soft and hard competitive learning).

nectionist literature in terms of the difficulty of defining a formal theory of what it means to generalize correctly (Hinton [29]). The inability to characterize a clustering of training patterns, like the inability to characterize a system's ability to generalize, is due to the absence of a suitable distance metric for training patterns.

Multiplicative Connections—A second feature our modular architecture has in common with systems previously proposed is the use of multiplicative connections. Because these connections allow one set of inputs to determine the mapping from the remaining inputs to the outputs of the system, they are particularly useful when there is a natural distinction between information to be processed and information that sets the context for processing. Specifically, multiplicative connections allow the activation of one unit of a network to act like the weight on the connection between two other units. Referring to Figure 3.2, if the activation of unit A is used as the weight on the connection between units B and C then the value of this weight is dependent on the input to A. An interesting special case occurs when the activation of A is binary. If the activation of A is zero then B has no influence on C. Alternatively, if the activation of A is one then the activation of B is the input to C. Many investigators have incorporated multiplicative connections in the design of their systems so that the system can compute different functions in different contexts (e.g., Hinton [27], Sejnowski [65], Feldman [20], Feldman and Ballard [21], McClelland [45], Maxwell, Giles, Lee, and Chen [44], Pomerleau [60], Yeung and Bekey [72]). Most relevant to our use of multiplicative connections is the work of Pollack [59] and Hampshire and Waibel [26].

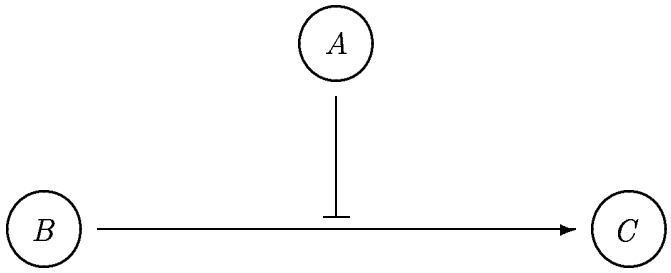

Figure 3.2: Multiplicative connection—The input to unit C is the product of the activations of units A and B.

The cascaded backpropagation (CBP) architecture of Pollack [59] consists of two networks called the context network and the function network. The output of the system is the output of the function network. However, the function network does not have a permanent set of weights. Instead, the outputs of the context network are used as the weights of the function network. Because the context network sets the weights of the function network, the mapping performed by the function network is said to be programmable by the context network. Using the chain rule, the backpropagation algorithm can be extended so that the appropriate weights of the context network can be learned. While the CBP architecture appears to consist of two networks, a simple change in notation shows that the architecture is equivalent to a single network with multiplicative connections. To the best of our knowledge, the CBP architecture was the first system to learn the appropriate weights on the multiplicative connections. Pollack has shown that the CBP architecture is capable of decomposing a task into a set of subtasks and of learning the set of subtasks faster than a single network can learn the undecomposed task. A disadvantage of this system is that the number of

output units of the context network must equal the number of weights of the function network and, thus, the number of units required by the system scales poorly with the size of the task. A second disadvantage is the the system uses the same network topology to perform all tasks.

Independently of the research described here, Hampshire and Waibel [26] developed an architecture similar to the one presented in Figure 3.1. They refer to this architecture as a "meta-pi" network. However, their system does not perform task decomposition. It is trained in two stages. In the first stage, a useful task decomposition is provided to the system, and each expert network is separately trained to perform one of the subtasks. In the second stage, the gating network is trained to switch in the appropriate expert network in the appropriate context. Because the meta-pi network does not itself perform task decomposition, the issues studied by Hampshire and Waibel are different from the issues we are addressing.

Stochastic Learning Automata—The gating network uses a learning procedure related to stochastic learning automata (Narendra and Thathachar [51]). Stochastic learning automata maintain a probability distribution over a set of actions. At each time step, an action is selected according to this distribution. If the environment provides the automaton with a reward, then the probability of performing the selected action is increased, whereas if the environment provides the automaton with a penalty, the probability of performing the selected action is decreased. After modifying the probability of performing the selected action, all the action probabilities are modified so that they sum to one. The learning procedure of the gating network is similar to

this, where the expert networks correspond to actions and the outputs of the gating network correspond to action probabilities. According to this view, if $J_{\mathbf{y}}(t) < \gamma\overline{J_{\mathbf{y}}}$, the gating network is rewarded; otherwise, it is penalized.

Brain Lateralization—The idea of competition between networks has appeared in the cognitive neuroscience literature in the form of the hypothesis that hemispheric specialization in humans is due to competition between neural subsystems. For example, Kosslyn [40] proposed that the brain contains many processing subsystems, each a neural network, which compete to learn about inputs. If the output of a network is used in subsequent computational processing, then the weights among connections in that network are altered so that the network produces the output faster and with less noise when the input recurs in the future. The weights of the networks whose outputs were not used in subsequent processing remain unchanged. Consequently, the networks compete to have their outputs used, and the strength of the training information received by a network is directly related to how that network fares in the competition. Kosslyn's credit assignment policy is thus nearly identical to the credit assignment policy of the modular architecture described above, and his hypothesis about why different subsystems of the brain learn to perform different tasks is consistent with our reasoning about why different expert networks of the modular architecture learn to perform different tasks.

Theories of brain lateralization also include the hypothesis that asymmetries in the cerebral hemispheres may influence the lateralization of brain functions (Geschwind

and Galaburda [24]). For example, if the left and right hemispheres compete for the ability to process language, then anatomical differences between the two hemispheres may bias the competition so that the left hemisphere usually wins. Similarly, a property of the modular architecture presented here is that the expert networks' architectures influence the competition between these networks. The architecture tends to allocate to each task the expert network whose topology is most appropriate for that task. For example, when required to perform a linear task and a nonlinear task, a linear expert network tends to win the competition to learn the linear task, whereas a nonlinear network tends to win the competition to learn the nonlinear task. A demonstration of this property is provided in Chapter 4 where we compare the performances of the modular architecture with that of two other networks trained with the backpropagation algorithm on the "what" and "where" vision tasks studied by Rueckl et al. [62].

3.5 An Alternative Error Function

The modular architecture consists of an arrangement of networks (Figure 3.1), an error function used in training the expert networks (Equation 3.2), and an error function used in training the gating network (Equation 3.6). Before reporting the results of training the modular architecture to perform various tasks, it is important to consider why the architecture was designed in this manner. In particular, we address the question of why the sum of squared error (SSE) between the desired

and actual outputs of the architecture (Equation 3.2) is used in training the expert networks but not in training the gating network. To answer this question, we compare the behavior of a modular architecture with that of a *cooperative architecture*, which is an architecture with the same arrangement of networks as the modular architecture but which uses the SSE in training both the expert and gating networks.

A cooperative architecture approximates a desired output pattern by linearly combining the outputs of all its expert networks. In contrast, a modular architecture tends to approximate a desired output pattern by "switching on" one expert network and "switching off" all other expert networks. The modular architecture may be considered a special case of the cooperative architecture in which the linear combination of the expert networks is restricted to a coefficient of one on the output of one of the expert networks and a coefficient of zero on the outputs of all other expert networks. Consequently, if cooperative and modular architectures have expert and gating networks with identical structures, then there may exist functions which the cooperative architecture can learn to approximate but which cannot be closely approximated by the modular architecture. Furthermore, if a solution is defined to be the set of functions computed by the expert networks at a single instant in time, then there may exist a large number of possible solutions that would allow a cooperative architecture to approximate a given function, whereas the set of solutions that would allow a modular architecture to approximate the same function may be relatively small. Because cooperative architectures have a larger number of potential solutions available

to them, one might be tempted to predict that these architectures learn faster than modular architectures.

Unfortunately, the question of which architecture learns faster cannot be answered in general, but rather its answer depends on such factors as the nature of the task that the architectures are required to perform, the structure of the expert and gating networks that compose the architectures, and the optimization procedures that are used to update the weights of the architectures. Consider a task in which different subtasks must be performed in different contexts. If the subtasks share many common properties, then an expert network that is useful in one context may also be useful in other contexts, albeit to a varying degree. In this case, a cooperative architecture may have an advantage over a modular architecture because it linearly combines the outputs of its gating networks using real–valued coefficients (however, see Chapter 5 for a modification to the modular architecture that allows it to explicitly model the common properties of a set of subtasks). Alternatively, if the subtasks do not share common properties or if the knowledge gained about one subtask interferes with the performance of other subtasks (e.g., negative transfer of training), then an expert network that is useful in one context is likely to be useless in other contexts or even diminish the performance of subtasks that must be performed in other contexts. In this case, a modular architecture may have an advantage over a cooperative architecture because it is constrained to combine the outputs of its expert networks using binary coefficients in which one coefficient is equal to one and all other coefficients

are equal to zero. Thus, it can quickly "switch off" an expert network that is not useful in a particular context.

The learning speeds of the modular and cooperative architectures are also dependent on the architectures' structures. In particular, whether the knowledge gained by an expert network during training on one subtask influences an architecture's ability to perform a second subtask is dependent on the structure of the architecture's expert and gating networks as well as the nature of the subtasks. If the expert and gating networks use relatively sparse representations (e.g., coarse–codes), then there is likely to be little generalization from one subtask to another. In contrast, if the networks use relatively compact representations, then there is likely to be alot of generalization from one subtask to another.

Whether the modular or cooperative architecture learns faster may also depend on the optimization procedure used to modify the architectures' weights. For example, one architecture may learn faster if the weights are modified using a conjugate gradient procedure with a line search, whereas the other architecture may learn faster if the weights are modified using a steepest descent procedure without a line search.

Although it is not possible to claim that one architecture learns faster than the other, we believe that many of the advantages of modular architectures over single networks (see Chapter 2) are also advantages of modular architectures over cooperative architectures. Specifically, suitably designed modular architectures are superior in terms of representation capabilities, generalization capabilities, and their ability to

satisfy constraints imposed by hardware limitations. These advantages stem from the fact that, due to a lack of computational constraints, a cooperative architecture's expert networks tend to learn functions that are arbitrarily related to the task that the architecture is trained to perform. In contrast, a modular architecture is sufficiently constrained so that its expert networks tend to learn functions that are meaningfully related to the target task.

In order to demonstrate this difference between the modular and cooperative architectures, we trained both architectures to approximate the function

$$f(x) = \begin{cases} \sin x & \text{if } x \in [-2\pi, 0] \\ \sin x + \sin 2x & \text{if } x \in (0, 2\pi]. \end{cases} \tag{3.7}$$

At each time step, x is selected from a uniform distribution over the interval $[-2\pi, 2\pi]$. The arrangement of networks used by the modular and cooperative architectures is shown in Figure 3.3. Each expert network is a linear unit with inputs set to $\sin x$ and $\sin 2x$. The gating network has two output units whose activations are between 0 and 1. The input to the gating network is a coarse–code of x (see the description given in Chapter 6 for more simulation details).

Because the modular architecture is constrained to "switch on" one expert network and "switch off" the other expert network at each time step, it can only compute f in two ways. The first expert network may compute $f(x) = \sin x$ and the second expert network may compute $f(x) = \sin x + \sin 2x$, or vice versa. Simulation results show that the modular architecture consistently finds one of these two possible solutions.

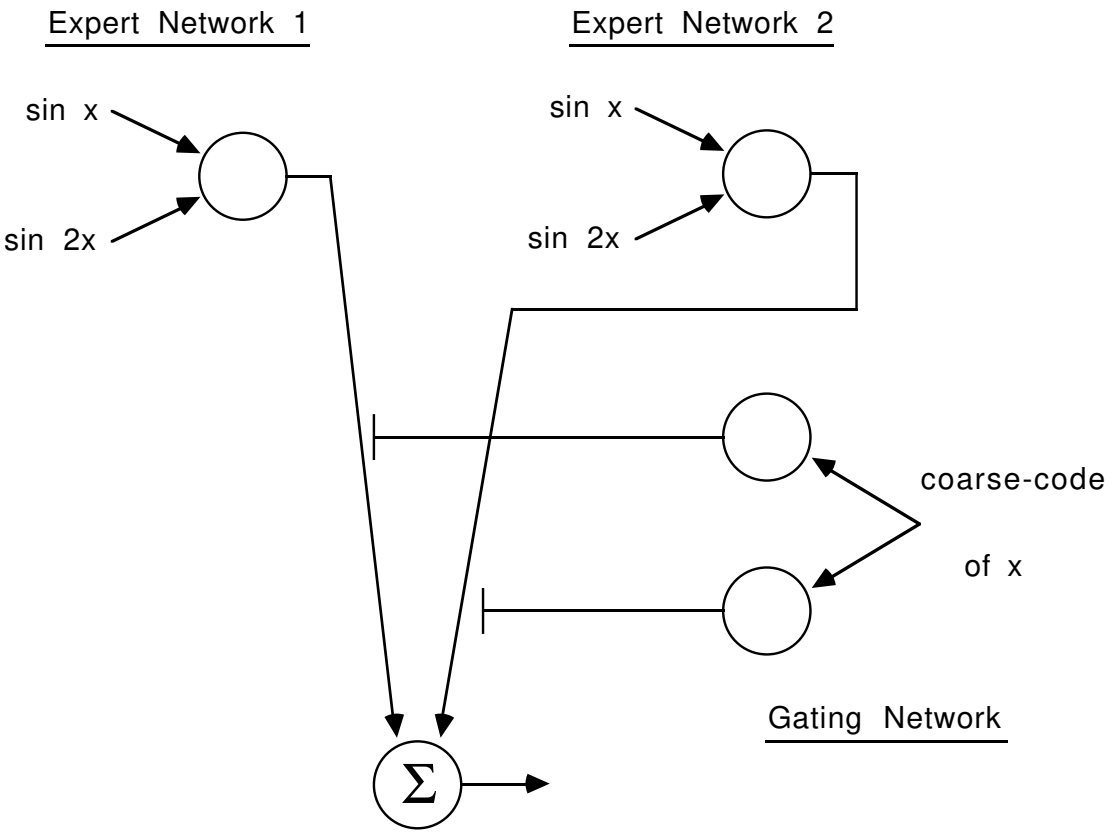

Figure 3.3: Arrangement of networks used by the modular and cooperative architectures.

The output units of the cooperative architecture's gating network may attain any value between 0 and 1. Consequently, there are an infinite number of ways in which the cooperative architecture can compute f. Not surprisingly, simulation results show that this architecture typically does not approximate f by computing $f(x) = \sin x$ with one expert network and $f(x) = \sin x + \sin 2x$ with the other expert network. Instead, the functions learned by the expert networks tend to have an arbitrary relationship with f, and both expert networks contribute to the output of the architecture for all values of x. Furthermore, each time the cooperative architecture is trained, the expert networks tend to learn to compute functions that are different from the functions that they learned during previous training sessions.

The expert networks of modular architectures, but not cooperative architectures, tend to learn identifiable and independent subtasks of the task that the architectures are trained to perform. Thus, modular architectures have advantages over cooperative architectures in terms of representation capabilities because they develop more interpretable representations. As argued in Chapter 2, systems with interpretable components are easier to understand, easier to extend by using the components as "building blocks" in the performance of more complex tasks, and can facilitate the use of attentional mechanisms. In addition, suitably designed modular architectures have advantages over cooperative architectures in terms of generalization capabilities. Modular architectures, but not cooperative architectures, can allocate expert networks with suitable structures to subtasks. Furthermore, for reasons analogous to those listed in Chapter 2, suitably designed modular architectures can more closely

satisfy several types of constraints imposed from the level of hardware than cooperative architectures. In conclusion, based on our discussion in Chapter 2 and in this section, we believe that the constraints placed upon the modular architecture give it significant advantages over single networks as well as other connectionist systems that use multiplicative connections, such as the cooperative architecture.

CHAPTER 4

THE WHAT AND WHERE VISION TASKS

Despite the fact that a variety of images are produced on the retina of a person watching a rotating or translating object, people recognize that the same object is depicted in each of the images. The ability to perform object recognition is said to be orientation and translation invariant. One hypothesis about how this invariance is achieved is that a canonical representation of each familiar object is stored, and the retinal image of an object is transformed so that the image and the representations can be compared. As a result of this transformation, information relevant to determining an object's spatial location is lost. This suggests that the process performing object recognition does not also perform spatial localization. Neuroscientists speculate that distinct cortical pathways of the primate visual system compute object recognition and spatial localization. Mishkin, Ungerleider, and Macko [49] reviewed evidence that a pathway running ventrally, interconnecting the striate, prestriate, and inferior temporal areas, computes object recognition, whereas a pathway running dorsally,

interconnecting the striate, prestriate and inferior parietal areas, computes spatial localization.

To investigate the computational advantages of employing distinct systems to perform these two tasks, Rueckl, Cave, and Kosslyn [62] compared the performance of two connectionist systems on an object recognition task (henceforth referred to as the "what" task) and a spatial localization task (henceforth referred to as the "where" task). The first system consisted of a single network that was required to perform both tasks. The second system consisted of two networks, one for each task. The retina was represented as a $5\,X\,5$ binary matrix. Each object was a specific pattern of binary entries in a $3\,X\,3$ matrix. At each time step of the training period, one of nine object matrices was centered at one of nine locations on the retinal matrix. The entries of the retinal matrix that lie outside the object matrix were set to zero. The "what" task is to identify the object; the "where" task is to identify its location.

The single–network system that Rueckl et al. [62] applied to these tasks was a network with two layers of modifiable weights. It had 25 input units, 18 hidden units, and 18 output units (see Table 4.1) and was strictly layered, meaning that all input units were connected to all hidden units, which in turn were connected to all output units. The input units encoded the $5\,X\,5$ retinal matrix, and the 18 output units corresponded to the nine possible objects and nine possible locations. The second system studied by Rueckl et al. [62] was a modification of this single–network system. Whereas the single–network system had connections from all hidden

Table 4.1: Systems studied by Rueckl, Cave, and Kosslyn on the "what" and "where" tasks.

System	Networks	Input	Output
1	$25 \rightarrow 18 \rightarrow 18$	retinal matrix	"what" and "where"
2	$25 \rightarrow 14 \rightarrow 9$	retinal matrix	"what"
	$25 \rightarrow 4 \rightarrow 9$	retinal matrix	"where"

units to all output units, the second system only had connections from the first 14 hidden units to the first 9 output units and from the remaining 4 hidden units to the remaining 9 output units (see Table 4.1). Thus, this system consisted of two separate networks, one for the "what" task and the other for the "where" task. Both the single–network system and the two–network system learned by means of the backpropagation algorithm.

Simulations showed that the two–network system learned the tasks faster and developed more interpretable representations than the single–network system. According to Rueckl et al. [62], both of these advantages resulted from the fact that the hidden units of the single–network system received inconsistent training information because they were connected to the output units for both the "what" and "where" tasks. Thus, the single–network system suffered from spatial crosstalk. In contrast, the hidden units of the two–network system did not receive inconsistent training information because they were connected to the output units for only one task. Based on these results, Rueckl et al. [62] concluded that it is better for a connectionist

system—and, by analogy, the primate visual system—to perform the "what" and "where" tasks in distinct networks.

An issue that Rueckl et al. [62] did not address, and the issue with which we are primarily concerned, is the development of a system that can learn for itself if it is better to decompose a learning task into two or more simpler tasks, and if so can allocate distinct networks for learning each simpler task. The next section presents simulation experiments using the "what" and "where" tasks to demonstrate that our modular architecture has this ability.

4.1 Simulation Experiments

Simulations of our modular architecture applied to the "what" and "where" tasks were conducted to investigate the architecture's ability to decompose learning tasks and the advantages that this ability may provide. For comparative purposes, several single networks were also simulated as applied to these tasks. Two sets of simulations were conducted. One set was designed to examine the modular architecture's behavior in the presence of temporal crosstalk; the other set of simulations addresses spatial crosstalk.

In describing the simulations, we refer to three different time periods: at each *time step*, a system is presented with a single input–output pair; during each *epoch*, a system is presented with every input–output pair in the training set exactly once; and a *run* consists of 100 epochs. In all simulations, the measure of a system's performance

is the percentage of input–output pairs that the system performs correctly during each epoch. A training pair is considered to be performed correctly when each output unit of the system has an activation greater than 0.6 when its desired activation is 1 and an activation less than 0.4 when its desired activation is 0.[1] The results for each system were averaged over 25 runs. 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 additional details of the simulations are provided in Appendix A.

4.1.1 Temporal Crosstalk

Recall that temporal crosstalk occurs when a system is trained to perform different tasks at different times. We trained a modular architecture and single networks to perform the "what" and "where" tasks. At any given time step, each system was trained to perform only one task. In addition to receiving 25 input values corresponding to the entries of the $5\,X\,5$ retinal matrix, each system received an input indicating whether it should perform the "what" or the "where" task. We call this input the *task bit*. There are 162 different input–output pairs (9 objects times 9 locations times 2 tasks).

[1]The more common sum of squared error measure was also computed and yielded results qualitatively similar to those produced using the percent correct. Thus, the sum of squared error is not reported here.

Two training procedures were employed to test each system's robustness in the presence of temporal crosstalk. One procedure avoids temporal crosstalk while the other does not. In the training procedure that avoids temporal crosstalk, called *random training*, at each time step an input pattern, consisting of a retinal matrix and a task bit, is randomly selected according to a uniform distribution over the set of 162 possible input patterns. The system's desired response is the object identity or location depending on the task bit. In the training procedure that is susceptible to temporal crosstalk, called *blocked training*, the task bit changes only once during each epoch. At the start of each epoch, one of the two tasks is randomly selected (with probability 0.5), and the input patterns presented during the first 81 time steps of the epoch have the task bit set to indicate the selected task. The input vectors presented during the last 81 time steps of the epoch have the task bit set to the other task. For example, during an epoch, a system may be required to perform the "what" task for 81 consecutive time steps followed by the "where" task for 81 consecutive time steps. At each time step within an epoch, the object and its location are randomly selected, and the desired system response depends on the task bit.

The first system trained to perform the "what" and "where" tasks is a single network trained with the backpropagation algorithm. It is strictly layered and contains 26 input units, 18 hidden units, and 9 output units (see Table 4.2). The input units encode the $5\,X\,5$ retinal matrix and the task bit. Figure 4.1 shows the learning curves for this network using the random training and blocked training procedures. The horizontal axis gives the number of epochs; the vertical axis gives the percent

Table 4.2: Systems used in the temporal crosstalk experiments.

Single Networks		
System	Networks	Input
1	$26 \rightarrow 18 \rightarrow 9$	retinal matrix and task bit
2	$26 \rightarrow 36 \rightarrow 9$	retinal matrix and task bit

Modular Architecture				
System	Expert Networks	Expert Networks' Input	Gating Network	Gating Network's Input
3	$26 \rightarrow 36 \rightarrow 9$	retinal matrix and task bit	$1 \rightarrow 3$	task bit
	$26 \rightarrow 18 \rightarrow 9$	retinal matrix and task bit		
	$26 \rightarrow 9$	retinal matrix and task bit		

of input-output pairs performed correctly. Clearly, this network learned faster with random training than with blocked training (at epoch 50, the difference between the performance with random and blocked training is statistically significant at the $p < 0.01$ level ($t = 4.84$)). This difference can be explained by the influence of temporal crosstalk in the case of blocked training. Temporal crosstalk attenuates the network's rate of learning in blocked training.

The second system trained to perform the "what" and "where" tasks is identical to the first system except that it has 36 hidden units instead of 18. Figure 4.2 shows the learning curves for this larger network. For the first 25 epochs, this network learned slightly faster with blocked training than with random training. However, for epochs 25–85, it learned significantly faster with random training than with

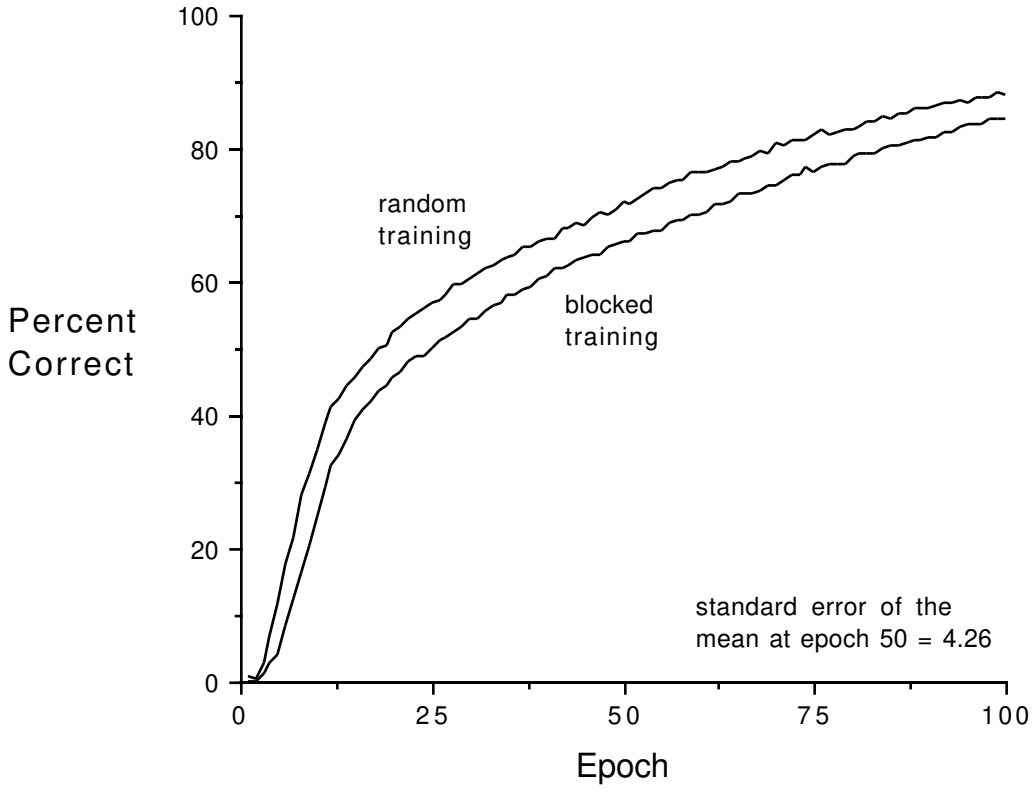

Figure 4.1: Learning curves for the $26 \rightarrow 18 \rightarrow 9$ network on the "what" and "where" tasks using random and blocked training.

blocked training (at epoch 50, the difference between the performance with random and blocked training is statistically significant at the $p < 0.01$ level ($t = 7.89$)). Similar to the $26 \rightarrow 18 \rightarrow 9$ network described above, temporal crosstalk attenuates the $26 \rightarrow 36 \rightarrow 9$ network's rate of learning in blocked training. Therefore, we cannot conclude that an abundance of hidden units makes a network more robust in the presence of temporal crosstalk.

The modular architecture that we applied to the "what" and "where" tasks, shown in Figure 4.3, consists of three expert networks and a gating network (see Table 4.2). Each expert network has 26 input units and 9 output units. The input units encode the $5\,X\,5$ retinal matrix and the task bit. Two of the expert networks are multi–layered networks with 36 and 18 hidden units respectively. The third expert network has a single layer, i.e., it has no hidden units. The gating network has 1 input unit which encodes the task bit and 3 output units corresponding to the three expert networks.

There are at least three ways that this modular architecture might successfully learn the "what" and "where" tasks. One of the multi–layer expert networks could learn to perform both tasks. In this case, the gating network must always gate on the network that learns both tasks and gate off the remaining expert networks. Although this is a possible solution, the results of Rueckl et al. [62] described above suggest that it is not the best solution in terms of learning speed and clarity of the resulting representation. A second possibility is that one of the multi–layer expert networks could learn the "what" task, and the other multi–layer expert network could learn the

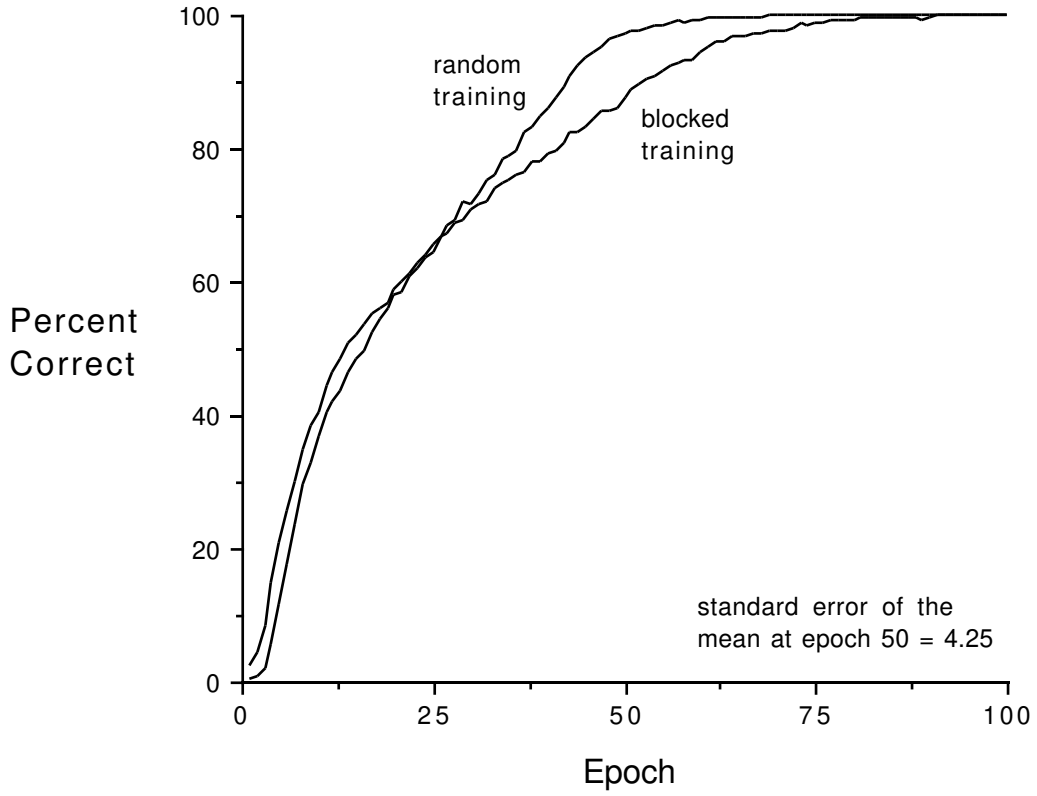

Figure 4.2: Learning curves for the $26 \rightarrow 36 \rightarrow 9$ network on the "what" and "where" tasks using random and blocked training.

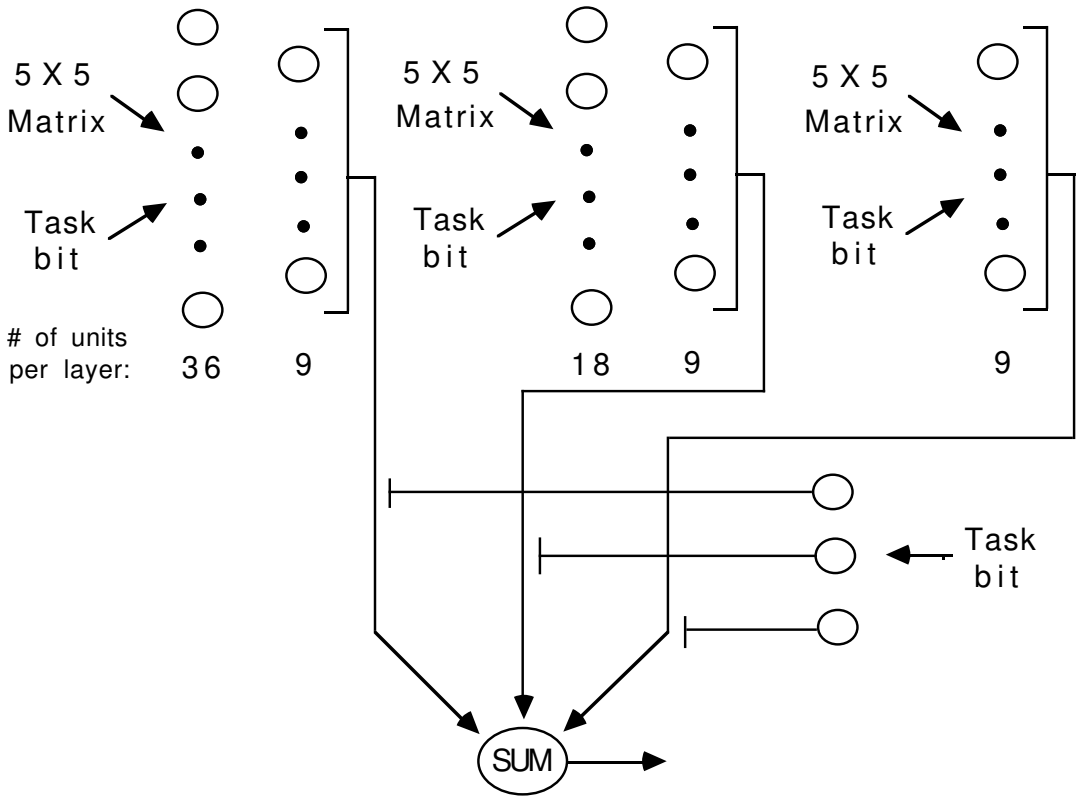

Figure 4.3: The modular architecture simulated in the temporal crosstalk experiments.

"where" task. In this case, the gating network must gate on the appropriate expert network based on the value of the task bit. This solution would indicate that the modular architecture had learned that it was required to perform two independent tasks and had allocated distinct networks to each task. However, a shortcoming of this solution is apparent when it is noted that, using the retinal images designed by Rueckl et al. [62], the "where" task is linearly separable. This means that the structure of the single–layer expert network most closely matches the "where" task. Consequently, a third and possibly best solution would be one in which one of the multi–layer expert networks learned the "what" task and the single–layer expert network learned the "where" task. This solution would not only show task decomposition but also the appropriate allocation of tasks to expert networks.

The simulation experiments using the modular architecture show that it produces this third possible solution. It always allocates the first multi–layer expert network to the "what" task and the single–layer expert network to the "where" task. This result suggests that, at least in some circumstances, the modular architecture is capable of performing function decomposition and that it tends to allocate to each function a network with a structure appropriate to that function. Learning curves for the modular architecture on the "what" and "where" tasks using random and blocked training are shown in Figure 4.4. Because little difference exists between performance with random and blocked training, these results suggest that the modular architecture is robust in the presence of temporal crosstalk (at epoch 50, the difference between the performance with random and blocked training is not statistically significant at

the $p < 0.01$ level ($t = 0.08$)). This robustness is due to the architecture's ability to allocate distinct networks to learn the different tasks.

A comparison of the learning curves in Figures 4.1, 4.2, and 4.4 shows that both the $26 \rightarrow 36 \rightarrow 9$ network and the modular architecture learn the "what" and "where" tasks faster than the $26 \rightarrow 18 \rightarrow 9$ network. In addition, the results suggest that of the three systems studied, only the modular architecture is capable of showing robust performance in the presence of temporal crosstalk.

4.1.2 Spatial Crosstalk

As presented so far, a limitation of the modular architecture is that only one expert network determines the output of the architecture at any one time step. When different tasks must be performed simultaneously, this system is unable to allocate different expert networks to learn the different tasks. Hence, the modular architecture is vulnerable to the detrimental effects of spatial crosstalk. Fortunately, a simple modification of the architecture overcomes this problem.

Consider the architecture illustrated in Figure 4.5. It consists of two expert networks and two gating networks. One gating network gates the first m components of each expert network's output vector, and the other gating network gates the remaining components. Suppose one function determines the desired values for the first m components of the architecture's output vector, $\mathbf{y}$, and a second function determines the desired values for the remaining components. The gating networks may allocate

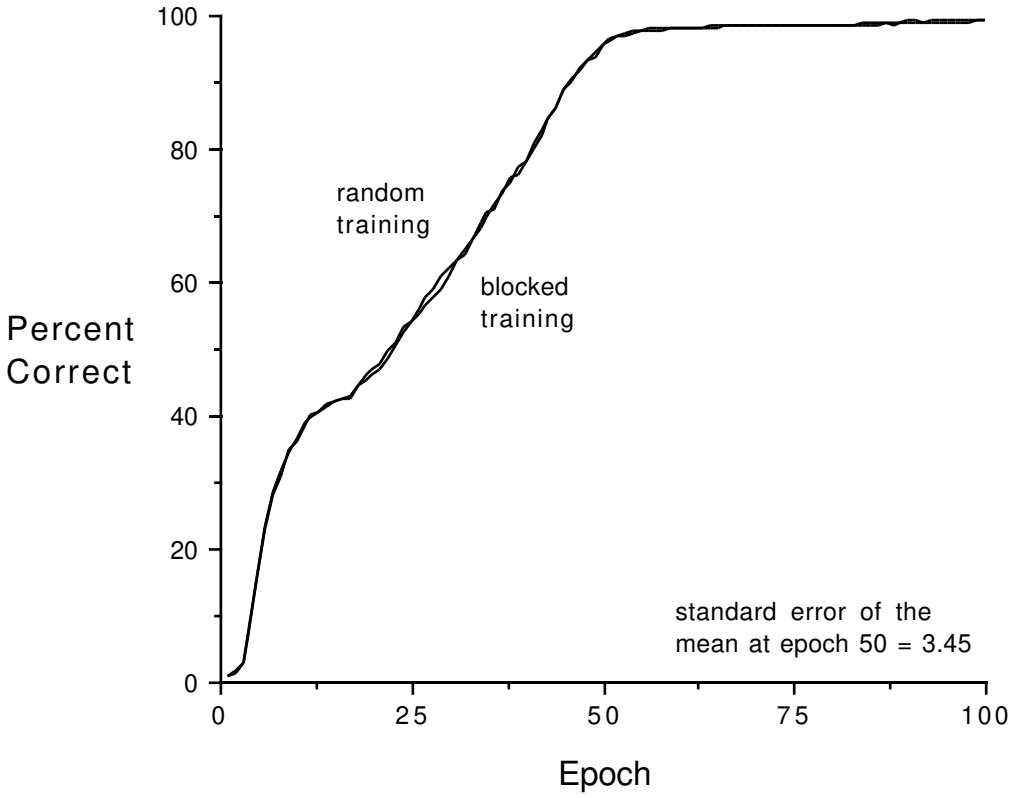

Figure 4.4: Learning curves for the modular architecture on the "what" and "where" tasks using random and blocked training.

the same expert network to both functions, or they may allocate different expert networks to each function.[2]

As in the case of the modular architecture with a single gating network, competition among the expert networks of the architecture with multiple gating networks determines how training patterns are allocated during learning. The learning algorithm for the modular architecture with multiple gating networks is identical to that described in Chapter 3 for an architecture with one gating network, with the proviso that the weight modifications are determined independently for each gating network.

Specifically, a modular architecture with multiple gating networks can be thought of as multiple separate modular architectures each having a single gating network. The separate architectures share expert networks although their gating networks gate

[2]Assuming that the output units of the gating networks produce binary outputs, the training of each expert network is identical to the case where an external teacher provides desired values for some output units of the expert network and places "don't care" conditions on other output units. At any given time step, let X denote the set of an expert network's output units whose outputs are multiplied by a gating network output of one, and let Y denote the set of output units whose outputs are multiplied by a gating network output of zero. During training, a non–zero error vector is backpropagated to the units in X, and the zero vector is backpropagated to the units in Y. Therefore, the output units in X, but not in Y, modify their weights and send error information to the hidden units of the network. In this sense, the training of this expert network is identical to the case where desired values are provided for the units in X and "don't care" conditions are placed on the units in Y (see Jordan [34] for a discussion of "don't care" conditions and spatial crosstalk).

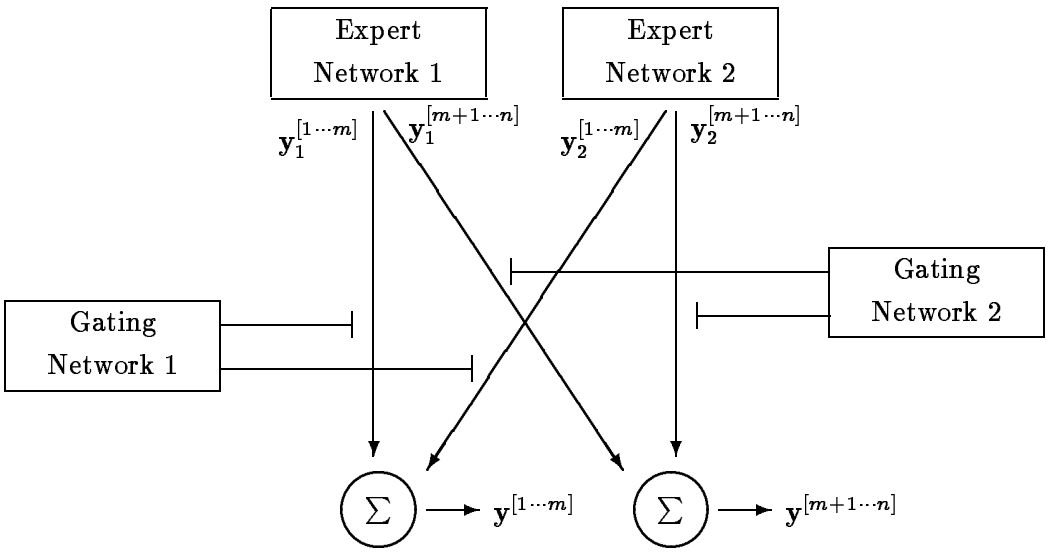

Figure 4.5: A modular architecture with multiple gating networks ($\mathbf{y}_1^{[1\cdots m]}$ denotes the vector whose components are the first m components of expert network 1's output vector, and the other expressions similarly denote subvectors).

different sets of the expert networks' output components. Consequently, the output components of the expert networks that are not gated by a gating network do not participate in the weight modification process for that gating network. This implies that for each gating network there is a separate process for determining when the performance of the architecture has significantly improved, and this process is identical to that used in architectures with single gating networks (Equation 3.4) except that it only depends on the error over the output components gated by the gating network. This results in different values λ_{WTA} and λ_{NT} for each gating network and therefore different error functions for each gating network. The error functions are given by Equation 3.6 with the substitution of the appropriate values for λ_{WTA} and λ_{NT}.

We trained a specific instance of the modular architecture shown in Figure 4.5 to perform simultaneously the "what" and "where" tasks. This means that at each time step the architecture's output pattern should correctly identify both the object and its location in the current input pattern. The specifications of this modular architecture are summarized in Table 4.3. It has two expert networks and two gating networks. Both expert networks have 25 input units and 18 output units. The input units encode the $5\,X\,5$ retinal matrix. Because both tasks are to be performed simultaneously, the architecture does not receive a task bit. The first 9 output units of each expert network encode the 9 possible objects, and the second 9 output units encode the 9 possible locations. The first expert network is a two–layer network with 36 hidden units, and the second expert network has a single layer of modifiable weights, i.e., it

Table 4.3: The modular architecture used in the spatial crosstalk experiment.

Expert Networks	Expert Networks' Input	Gating Networks	Gating Networks' Input
$25 \rightarrow 36 \rightarrow 18$	retinal matrix	$0 \rightarrow 2$	none
$25 \rightarrow 18$	retinal matrix	$0 \rightarrow 2$	none

has no hidden units. The two gating networks are extremely simplified: each has two output units but no hidden units and no input units. Each gating network therefore just consists of two output units, each with a single bias weight. The first gating network gates the first 9 output components of the expert networks, and the second gating network gates the second 9 output components of the expert networks.

There are two ways that this modular architecture might successfully learn to perform the "what" and "where" tasks. One possibility is that the multi–layer expert network learns to perform both tasks, and the second possibility is that the multi–layer expert network learns to perform the "what" task whereas the single–layer expert network learns to perform the "where" task. For the reasons described above, the second possibility is the better solution.

Our first simulations of the architecture on the two tasks produced disappointing results. The architecture consistently allocated the single–layer expert network to both tasks. Because the "what" task is not linearly separable, the architecture did not correctly perform this task. Further analysis of the modular architecture's learning

rule reveals the reasons for this behavior. Recall from Section 3 that the architecture's current performance is compared with its past performance at each time step. If the performance has significantly improved, then the weights of a gating network are modified so that the gating network output corresponding to the winning expert network increases toward one, and the outputs corresponding to the losing expert networks decrease toward zero. If the current performance does not show significant improvement, then a gating network's weights are modified so that all of its outputs are moved towards a neutral value.

In the initial training runs, it turned out that the single–layer expert network approximated the "what" function more quickly than did the multi–layer expert network. Therefore, the output of the first gating network corresponding to the single–layer expert network approached one, and the output corresponding to the multi–layer expert network approached zero. However, because the "what" task is not linearly separable, the performance of the single–layer expert network on the "what" task could not improve past a certain low level.[3] Consequently, we expected the outputs of the gating network to eventually approach the neutral value and then change so as to appropriately reallocate tasks to expert networks, but this did not happen. The architecture failed to reallocate tasks because even the low level of performance achieved by performing both tasks with the single–layer expert network was

[3]At best, the single–layer expert network can correctly perform 54 percent of the "what" task's input–output pairs.

better than the performance attainable by nearby weight values. The architecture consistently became trapped in this local error minimum.

Rather than devising a mechanism to allow the architecture to escape from this kind of local minimum, we modified the learning process in order to make the architecture less likely to become trapped in this way. During training we varied 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 term in Equation 3.6, contains the factor λ_{NT} which is non–zero only when the architecture's performance has not significantly improved.[4] Instead of setting λ_{NT} for a gating network to one when performance has significantly improved, we initialized it to a value greater than one at the start of training and slowly decreased it to one during training.[5] Larger values of λ_{NT} increase the tendency of the gating network outputs to remain near the neutral value and therefore prolong the period of training before the expert networks specialize. This gives 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. As a result, an expert network that learns a task quickly at

[4]Recall that when there are multiple gating networks, the case we are discussing, there is a different λ_{NT} for each gating network.

[5]The effect of altering λ_{NT} is to alter the step size of changes in gating network weights that occur when the architecture's performance has not significantly improved.

first, such as the single–layer network in the "what" task, does not necessarily become allocated to that task.

When modified in this manner, the modular architecture with two gating networks that we simulated consistently learned to allocate the multi–layer expert network to the "what" task and the single–layer expert network to the "where" task. As the outputs of the expert networks came to be gated in an appropriate manner by the gating networks, the degree of conflict in the training information received by the output units of the expert networks decreased, that is, spatial crosstalk decreased. However, we cannot report that as a result of decreased spatial crosstalk the modular architecture learned the "what" and "where" tasks more rapidly than did the single–network system studied by Rueckl et al. [62]. It is not surprising that prolonging the period before the expert networks specialize considerably slows learning, but this may be necessary for the modular architecture to selectively allocate networks to tasks based on the suitability of the networks' topologies.

In summary, we have studied the performances of single networks and modular architectures on the "what" and "where" tasks of Rueckl et al. [62]. Our major goal has been to demonstrate the modular architecture's ability to perform function decomposition in the sense of learning to allocate different networks to learn different functions. A second goal has been to suggest that the architecture tends to allocate to each function a network with a topology that is appropriate to that function. We have also shown that, at least in the case presented here, the modular architecture is robust in the presence of temporal crosstalk.