Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks.*

Robert A. Jacobs[1]

Michael I. Jordan[2]

Andrew G. Barto[1]

[1]Department of Computer & Information Science

University of Massachusetts, Amherst, MA 01003

[2]Department of Brain & Cognitive Sciences

Massachusetts Institute of Technology, Cambridge, MA 02139

COINS Technical Report 90–27

March 1990

---

*We are extremely grateful to Kyle Cave, Geoffrey Hinton, Stephen Kosslyn, Michael Mozer, and Steven Nowlan for sharing their thoughts and suggestions in regard to the material in this paper. This material is based upon work supported by the Siemens Corporation, the Air Force Office of Scientific Research, Bolling AFB, under Grant AFOSR-89-0526, and the National Science Foundation under Grant ECS-8912623.

Abstract

A novel modular connectionist architecture is presented in which the networks composing the architecture compete to learn the training patterns. An outcome of the competition is that 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. The architecture's performance on "what" and "where" vision tasks is presented and compared with the performance of two multi–layer networks. Finally, it is noted that function decomposition is an underconstrained problem and, thus, different modular architectures may decompose a function in different ways. We argue that a desirable decomposition can be achieved if the architecture is suitably restricted in the types of functions that it can compute. Appropriate restrictions can be found through the application of domain knowledge. A strength of the modular architecture is that its structure is well–suited for incorporating domain knowledge.

Although many biologists and psychologists share the view that the brain has a modular architecture, there is no general agreement on the number of modules, the function of the modules, the nature of the interaction between modules, or the manner in which the modules develop. One reason for this diversity of opinion is that answering questions about the modular nature of the brain involves the difficult task of reasoning about a system with a large number of interacting components. Even systems of interacting components with a small fraction of the brain's complexity present formidable conceptual and analytical difficulties. In many cases, mathematical and computer models provide essential tools for understanding aspects of these systems. One class of models that has the potential for helping to answer questions about modular systems is the class of connectionist models, also known as artificial neural network models.

A hierarchical classification of the components of connectionist models may be defined in which a unit is the finest level of classification, a layer is a coarser level, and a network is a still coarser level. Connectionist researchers typically design systems that are modular at the level of units or layers. In this paper we argue that there are significant practical and theoretical advantages to be realized by considering modularity at the level of networks. In other words, we argue that connectionist architectures should consist of multiple networks, and that connectionist learning algorithms should be designed to take advantage of this modular structure.

Although terms such as *layer* or *network* are imprecise, it is generally agreed that they provide a convenient language for discussing connectionist architectures. An analogous situation occurs in the neurosciences where researchers debate whether nervous systems are

best conceptualized at the level of the neuron or at coarser levels such as the column, hypercolumn, or area (Kaas [30]). This debate persists despite the lack of agreement on the precise definitions of the coarser taxonomic levels. For our purposes, we rely on the intuition that portions of an architecture larger than a unit or layer that learn and perform different functions constitute different networks.

This paper introduces a novel modular connectionist architecture in which the networks composing the architecture compete to learn the training patterns. An outcome of this competition is that 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. An additional property of this architecture is that it tends to allocate to each task the network whose topology is most appropriate to that task. This property has implications for both the efficiency of learning and the forms of generalization the modular architecture produces.

The paper is organized as follows. Section 1 discusses computational advantages of modular connectionist architectures that are capable of allocating different networks to compute different functions. Section 2 describes the modular architecture that we have developed. Section 3 describes the "what" and "where" vision tasks studied by Rueckl, Cave, and Kosslyn [46]. Section 4 compares the performance of our modular architecture with that of two networks using the backpropagation algorithm (le Cun [33], Parker [40], Rumelhart, Hinton, and Williams [47], Werbos [52]) on the "what" and "where" vision tasks. Section 5 discusses the relationship between the structure of the modular architecture and the nature of the

task decomposition discovered by the architecture.

1 Advantages of Modular Connectionist Architectures

Theorists have shown that connectionist networks are universal approximators, meaning that for any given function there is a connectionist network capable of approximating it arbitrarily closely (e.g., Hornik, Stinchcombe, and White [25]). Given this result, what advantages might a modular architecture consisting of several connectionist networks have over a single network? In this section we answer this question by arguing that modular architectures have advantages in terms of learning speed, generalization capabilities, representation capabilities, and their ability to satisfy constraints imposed by hardware limitations. This discussion is not specific to the architecture that we have developed; it applies to the general class of multi–network systems of which ours is an example.

1.1 Learning speed

Several characteristics of modular architectures suggest that they should learn faster than single connectionist networks. One such characteristic is that modular architectures can take advantage of *function decomposition*. If there is a natural way to decompose a complex function into a set of simpler functions, then a modular architecture should be able to learn the set of simpler functions faster than a single network can learn the undecomposed complex

function. For example, consider the absolute value function

$$f(x) = \begin{cases} -x & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases} \tag{1}$$

This nonlinear function can be learned by a single network with at least one layer of hidden units. Alternatively, it can be learned by a modular architecture consisting of two networks, each a single linear unit, and a mechanism for switching on the appropriate network in the appropriate context. One network can learn the function $f(x) = x$, and the other network can learn the function $f(x) = -x$. Assuming that it is relatively easy to learn the switching mechanism, the modular architecture should be able to learn faster than the single network because it does not use hidden units and each module only needs to learn a linear function.

In addition to being able to take advantage of function decomposition, modular architectures can be designed to reduce the presence of conflicting training information that tends to retard learning. We refer to conflicts in training information as *crosstalk* and distinguish between spatial and temporal crosstalk. Spatial crosstalk occurs when the output units of a network provide conflicting error information to a hidden unit. Jordan [28] and Plaut and Hinton [41] noted that this occurs when the backpropagation algorithm is applied to a single network containing a hidden unit that projects to two or more output units. For example, suppose that a hidden unit projects via positive weights to two output units and that when compared to the desired output values, the output level of the first output unit is too small, whereas the output level of the second output unit is too large. Using the backpropagation algorithm, the first output unit provides derivative information specifying that the hidden unit should have a larger output. However, the second output unit provides derivative in-

formation specifying that this same hidden unit should have a smaller output. This conflict in derivative information is an instance of spatial crosstalk.[1]

Plaut and Hinton [41] noted that a modular architecture consisting of a separate network for each output unit is immune to spatial crosstalk. For example, consider the systems shown in Figure 1. Panel A shows a single network and Panel B shows a modular architecture consisting of three separate networks, one for each output unit. Although these systems can be applied to the same tasks, the modular architecture is immune to spatial crosstalk because each hidden unit projects to a single output unit.

In contrast to spatial crosstalk—where a unit receives inconsistent training information at a single instant in time—a unit might receive inconsistent training information at different times, a situation resulting in temporal crosstalk. One form of temporal crosstalk occurs when a network is trained to perform different functions at different times. For example, suppose that when a network is trained to perform one function, some of its hidden units become particularly useful in performing that function. When this same network is later trained to perform a second, different function, one would like it to learn the second function without unnecessarily degrading its performance on the first function. This can be accomplished if the hidden units that participate in implementing the second function are not the ones that implement the first function. However, as Sutton [50] observed, gradient descent

[1] Although spatial crosstalk is clearly seen in terms of the backpropagation algorithm, it is not limited to networks trained using this algorithm. A network trained using any algorithm that approximates gradient descent (e.g., the A_{R-P} algorithm of Barto and Anandan [7] and Barto and Jordan [8]) can suffer from spatial crosstalk.

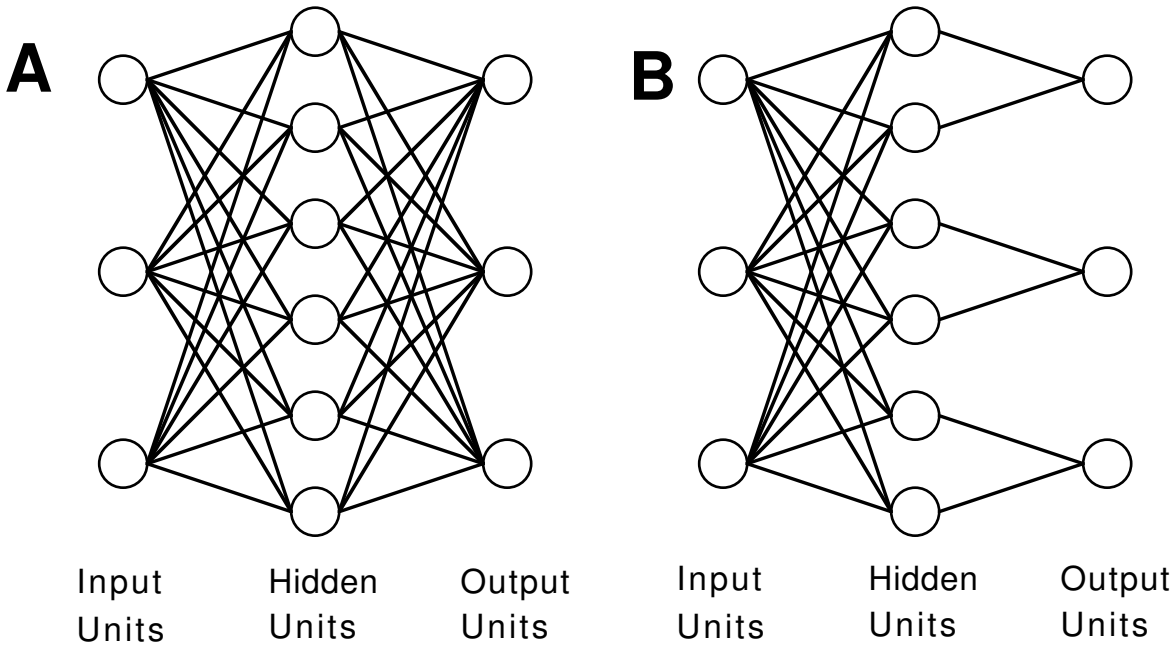

Figure 1: A: A network that is susceptible to spatial crosstalk. B: A modular architecture that is not susceptible to spatial crosstalk.

algorithms produce the opposite behavior because they preferentially modify the weights of the hidden units that are already useful. Therefore, after training on the second function, the network's performance on the first function will tend to be significantly degraded.

Temporal crosstalk arises in another situation closely related to that just described: when training patterns come from one region of the input space for many consecutive trials and later come from a different region for many consecutive trials. [2] If the function has different properties in these different regions, the network would receive conflicting training information. Training with patterns in the second region of the input space can result in degradation of the network's ability to perform correctly for patterns in the first region.

The conflicts in training information producing spatial and temporal crosstalk can be thought of in terms of transfer of training. Training a system to compute one function may facilitate the system's ability to learn a second function, a situation referred to as positive transfer of training, or it may retard the system's ability to learn a second function, a situation referred to as negative transfer of training. Suitably designed modular architectures should learn faster than single networks because similar functions can be learned by the same network of the modular architecture, resulting in the benefits of positive transfer of training, whereas dissimilar functions can be learned by different networks, thereby avoiding the detrimental effects of negative transfer of training.

[2] Although this is not how one would typically prefer to train a network, it may be unavoidable. For example, in training a network to control a dynamical system, the state of the dynamical system is likely to change slowly relative to the rate at which the system's variables are observed. Hence, the network is likely to receive inputs from the same region of the input space for many consecutive time steps.

1.2 Generalization

Unless the structure of a network is well–matched to the function on which it is trained, it is unreasonable to expect the network to generalize well (e.g., Denker et al. [11]). For example, although both a single–layer linear network and a multi–layer nonlinear network can accurately learn a set of training pairs generated by a linear function, one would expect the single–layer network to generalize better because its structure is closer to the structure of the function being learned. If a modular architecture were able to decompose a complex function into a set of simpler functions and allocate an appropriately structured network to each simpler function, then one would expect good generalization. In this case, the mechanism responsible for allocating patterns to the networks of the modular architecture would be automating part of the process of matching network structures to tasks, the other part of the process being played by the initial choice of the modular architecture's repertoire of networks.

A second reason that a modular architecture should generalize better than a single network involves the difference between local and global generalization. Global generalization occurs when the learning of a training pattern from one region of the input space influences the network's performance on patterns from a much wider region of the input space. As suggested above in the discussion of temporal crosstalk, when the function generating the training patterns possesses different characteristics in different regions, global generalization is an undesirable property due to negative transfer of training. In contrast, modular architectures perform local generalization in the sense that each network of the architecture only learns patterns from a limited region of the input space. Therefore, training a modular

architecture on a training pattern from one of these regions does not affect the architecture's performance on patterns from the other regions.

1.3 Representation

Modular architectures tend to develop representations that are more easily interpreted than the representations developed by single networks. By this we mean that it tends to be easier to understand how a modular architecture implements a function than it is to understand how a single network implements the same function. This property was demonstrated by Rueckl et al. [46] who trained two connectionist systems to perform object recognition and spatial localization from simulated retinal images. As a result of learning, the hidden units of the system using separate networks for the recognition and localization tasks contribute to the solutions of these tasks in more understandable ways than the hidden units of the single network applied to both tasks. In the former system, a different set of hidden units is used to represent information about the different tasks. In the latter system, on the other hand, the same set of hidden units is forced to represent information about both tasks despite the fact that the two tasks are relatively independent.

In addition to making it easier for experimenters to understand their systems, interpretable representations may make it easier for one portion of a system to understand the tasks performed by other portions. Minsky [37] argued that if one module can determine the task that is performed by a second module, then the former module can request the latter module to perform that task in the appropriate situation. This view emphasizes that the

functions learned by the modules can be thought of as building blocks to be used on other occasions in the performance of more complex tasks.

Another advantage of using different networks to perform different portions of a task is that this representation facilitates the use of attentional mechanisms. Cowey [10] speculated that properties of a visual image may be attended to or unattended to through the operation of a relatively coarse mechanism that enhances or suppresses the activations of entire populations of processing units. Only if different modules of an architecture represent different portions of a task would we generally expect such mechanisms to have coherent and sensible effects.

Other reasons that modular architectures are superior to single networks in their representational capabilities are related to familiar arguments advanced in the computer science and psychology literatures: because modular architectures localize functions and develop more interpretable representations, they are easier for researchers to debug; because much of human knowledge is modularized, it is easier for researchers to embed domain knowledge into a connectionist system when the system is organized in a modular fashion; and because modules can be added one at a time, modular architectures facilitate the development of connectionist systems in an incremental manner.

1.4 Hardware constraints

Another property of modular architectures suggesting advantages over single networks is that modular architectures can more closely satisfy several types of constraints imposed at

the level of hardware implementation. In particular, suitably designed modular architectures can reduce the number of units and the lengths of connections.

In a discussion of representations employed by the brain, Ballard [3] suggested that a limitation on the number of neurons compels the brain to adopt a modular architecture. He hypothesized that the brain uses a coarse–code (Albus [1], Hinton [23]) to represent multi–dimensional spaces. Using this kind of representation, the number of neurons required to represent a space is $\frac{N^k}{D^{k-1}}$, where k is the dimension of the space, N is the number of just–noticeable differences in each dimension, and D is the diameter of the receptive field of each neuron (Hinton [23], Ballard [3]). Because this rapid growth in the number of neurons limits the number of dimensions that can be represented in a cortical area, high–dimensional spaces must be represented in such a way that different dimensions are represented in different areas. Because different areas represent different dimensions, these areas must compute different functions. Analogously, in order to reduce the number of units required by a connectionist system, one may distribute the representation of multi–dimensional spaces among multiple networks.

Other speculations as to why the brain contains multiple cortical areas are also relevant to connectionist systems. Cowey [9] suggested that if the cortex uses lateral inhibition to sharpen various visual attributes, such as edges, orientation, color, disparity, spatial frequency, size, and movement, then a retinotopic representation of these attributes allows the use of relatively short connections between neurons since this representation places the interneurons necessary for receptive–field tuning relatively close together (Durbin and Mitchi-

son [13]). If neurons for all the different attributes are represented in a single retinotopic map, however, the connections needed to sharpen the tuning of individual neurons for one of these attributes would be unnecessarily long. The existence of multiple retinotopic maps, each in a separate cortical area, allows the cortex to highlight several attributes using short local connections.

Similarly, Barlow [5] suggested that in order to detect various visual attributes using neurons whose connections are of minimal length, the cortex must employ multiple representations. He argued that because the brain primarily contains local connections, it is difficult, using a retinotopic representation, to detect similarities among non–contiguous locations of the visual field. In order to detect such similarities, it is necessary to map the information in the retinal image so that similar events are represented close to each other independently of the retinal coordinates of the events. For example, using a retinotopic representation and neurons with local connections, it is difficult to detect the co–linearity of line segments located at different places in the retinal image. However, in a different, non–retinotopic representation (e.g., the Hough transform (Duda and Hart [12], Ballard [2])) nearly co–linear line segments can be represented by neighboring neurons. In general, the existence of multiple representations, each in a separate area, allows the brain to detect several attributes using short connections.

The hypotheses of Ballard [3], Cowey [9], and Barlow [5] outlined above about why the cortex employs multiple representations located in different cortical areas are also relevant to connectionist architectures. These hypotheses suggest that in order to evaluate several attributes using units with local connections, connectionist systems should employ multiple

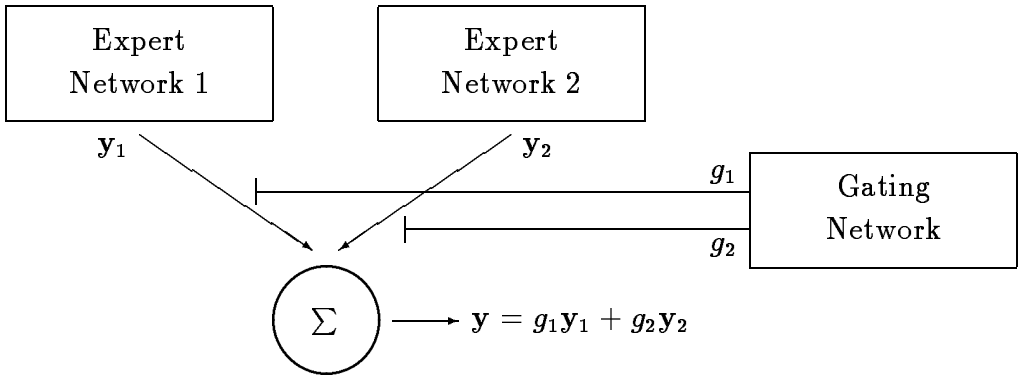

Figure 2: A modular connectionist architecture.

representations located in different networks.

2 A Modular Connectionist Architecture

In this section 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.

2.1 Output of the Architecture

The architecture illustrated in Figure 2 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 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

$$\mathbf{y} = \sum_{i=1}^{n} g_i \mathbf{y}_i. \tag{2}$$

2.2 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

$$J_\mathbf{y} = \frac{1}{2}(\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y}). \tag{3}$$

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:

$$\overline{J_{\mathbf{y}}}(t) = \alpha J_{\mathbf{y}}(t) + (1-\alpha)\overline{J_{\mathbf{y}}}(t-1), \tag{4}$$

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,

$$\begin{aligned} &\text{If} \qquad J_{\mathbf{y}}(t) < \gamma\overline{J_{\mathbf{y}}}(t-1), \qquad (5)\\ &\text{Then } \lambda_{WTA} = 1 \text{ and } \lambda_{NT} = 0\\ &\text{Else } \lambda_{WTA} = 0 \text{ and } \lambda_{NT} = 1, \end{aligned}$$

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

$$J_{\mathbf{y}_i} = \frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_i). \tag{6}$$

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:

$$
\begin{aligned}
J_G &= \lambda_{WTA}\frac{1}{2}\sum_{i=1}^{n}(g_i^* - g_i)^2 + \\
&\lambda_{WTA}\frac{1}{2}(1 - \sum_{i=1}^{n} g_i)^2 + \\
&\lambda_{WTA}\sum_{i=1}^{n} g_i(1 - g_i) + \\
&\lambda_{NT}\frac{1}{2}\sum_{i=1}^{n}(\frac{1}{n} - g_i)^2.
\end{aligned}
\tag{7}
$$

Due to the definition of λ_{WTA} and λ_{NT} (Equation 5), the first three terms of Equation 7 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 $\frac{1}{n}$.

2.3 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 2); 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 2, 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 backpropated 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.

2.4 Relationship to Previous Research

Competitive Learning—A common feature of our modular architecture and some systems previously proposed is the use of competitive learning. There are many unsupervised learning algorithms that make use of competition between the units of a network (e.g., Kohonen [31], Rumelhart and Zipser [48], Grossberg [20], Reggia [45], Durbin and Willshaw [14], Yuille and Grzywacz [55]). In all of these systems, the units of a single network compete for the right to respond maximally to a given subset of input patterns. The competition is based on the relative amount of match between the input vector and each weight vector of the network. As a result of this competition, different units learn to respond to patterns from different portions of the input space.

A distinguishing feature of the competitive aspects of the architecture presented here is that the objective is not to cluster the input patterns into natural groupings but rather to cluster the training patterns, which are input patterns together with desired output patterns, into natural groupings. The competition is not between different units of a network but rather is between different networks of the modular architecture. The expert networks compete for the right to learn a given subset of training patterns. As a result of this competition, different expert networks learn different training patterns and, thus, learn to compute different functions. A second distinguishing feature of the architecture is that the competitive process is supervised; the competition is based on the abilities of the expert networks to approximate the desired output values.

Multiplicative Connections—A second feature our modular architecture has in common with

systems previously proposed is its use of multiplicative connections. 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 [22], Sejnowski [49], Feldman [15], Feldman and Ballard [16], McClelland [36], Maxwell, Giles, Lee, and Chen [35], Pomerleau [44], Yeung and Bekey [54]). Most relevant to our use of multiplicative connections is the work of Hampshire and Waibel [21] and Pollack [43].

Independently of the research described here, Hampshire and Waibel [21] developed an architecture similar to the one presented in Figure 2. 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.

Pollack [43] developed a system, called the cascaded backpropagation (CBP) architecture, that uses multiplicative connections in order to learn to compute different functions in different contexts. 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. Disadvantages of this system are that the number of units required by the system scales poorly with

the size of the task and the system uses the same network topology to perform all tasks.

Stochastic Learning Automata—The gating network uses a learning procedure related to stochastic learning automata (Narendra and Thathachar [39]). 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 [32] 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 [19]). 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 Section 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. [46].

3 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 [38] 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 et al. [46] 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

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"

Table 1: Systems studied by Rueckl, Cave, and Kosslyn on the "what" and "where" tasks.

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. [46] 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 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 $5\,X\,5$ matrix, and the 18 output units corresponded to the nine possible objects and nine possible locations. The second system studied by Rueckl et al. [46] was a modification of this single–network system. Whereas the single–network system had connections from all hidden 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 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. [46], 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. [46] 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. [46] 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 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.[3] 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 the appendix.

4.1 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

[3]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.

$5\,X\,5$ 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).

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 2). The input units encode the $5\,X\,5$ retinal matrix and the task bit. Figure 3 shows the learning curves for this network using the

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		

Table 2: Systems studied in temporal crosstalk experiments.

random training and blocked training procedures. The horizontal axis gives the number of epochs; the vertical axis gives the percent 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 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 sig-

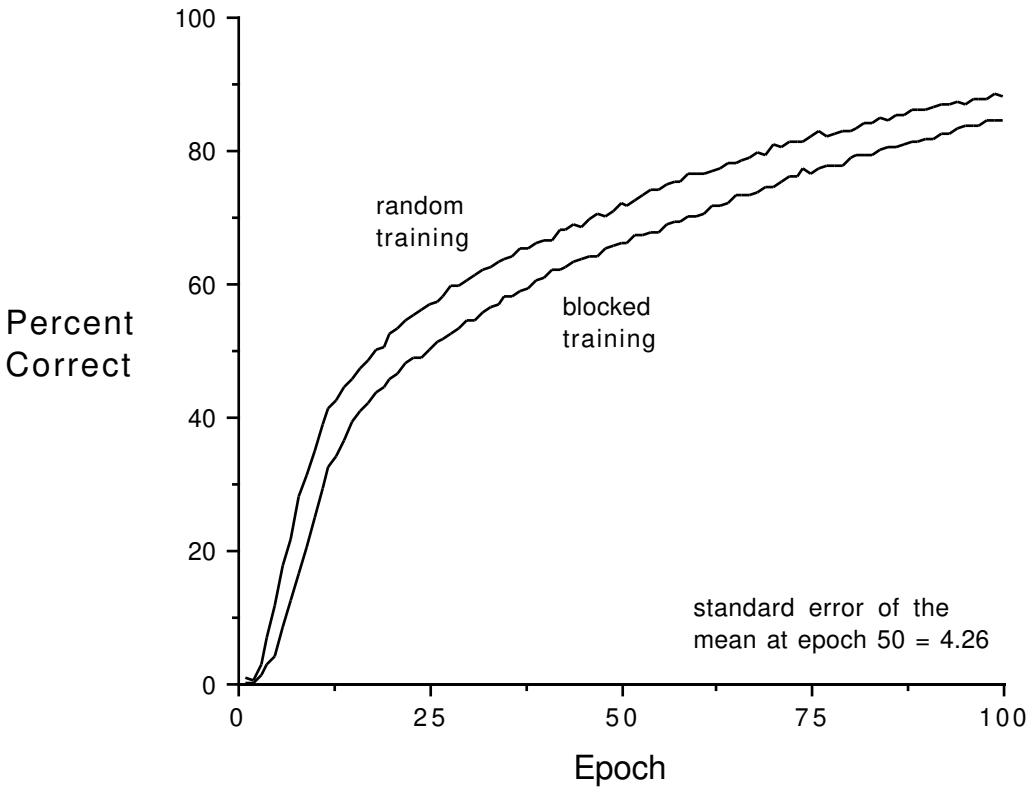

Figure 3: Learning curves for the 26 $\rightarrow$ 18 $\rightarrow$ 9 network on the "what" and "where" tasks using random and blocked training.

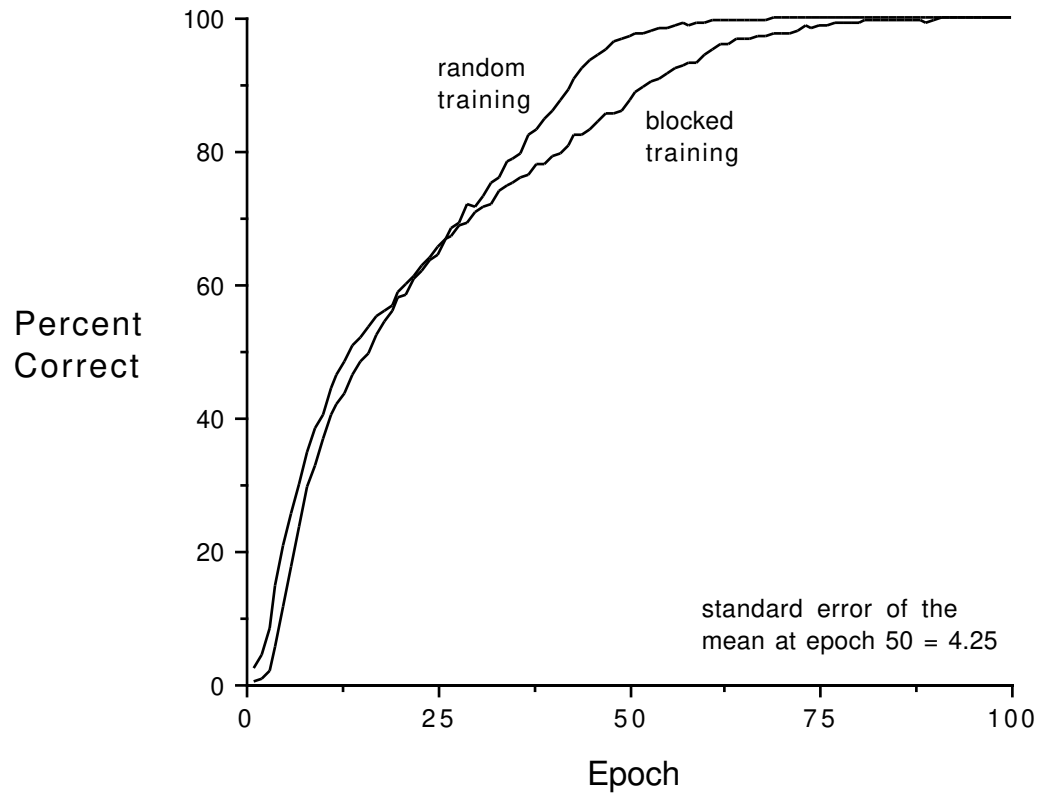

Figure 4: Learning curves for the 26 → 36 → 9 network on the "what" and "where" tasks using random and blocked training.

nificantly 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 ($t = 7.89$)). Similar to the 26 → 18 → 9 network described above, temporal crosstalk attenuates the 26 → 36 → 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

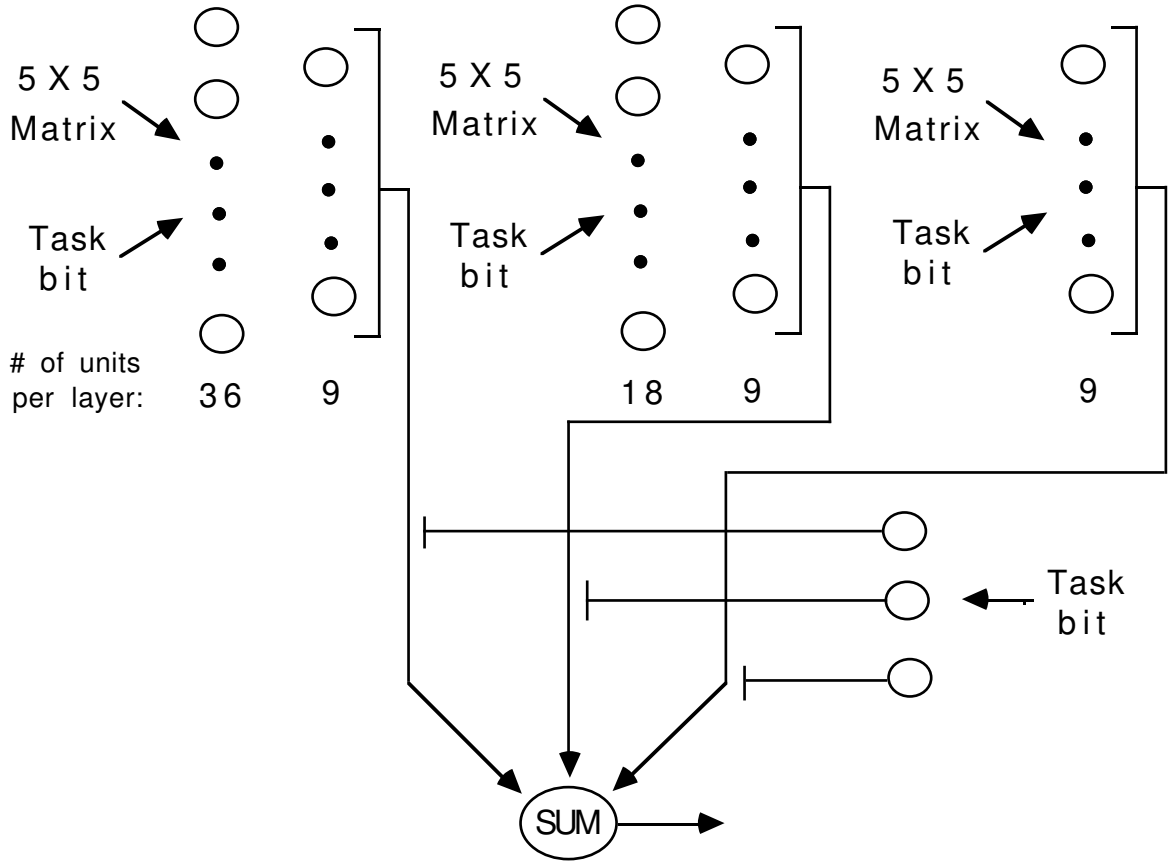

Figure 5: The modular architecture simulated in the temporal crosstalk experiments.

Figure 5, consists of three expert networks and a gating network (see Table 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. [46] 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 "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. [46], 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 6. 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 3, 4, and 6 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.

4.2 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.

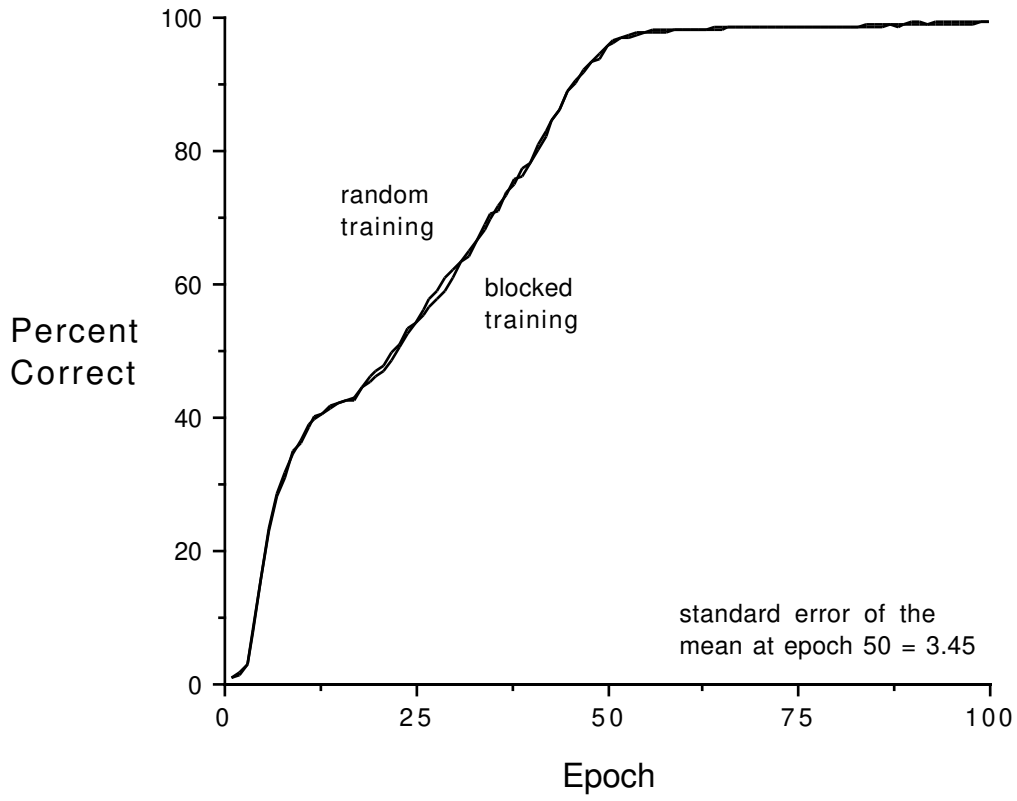

Figure 6: Learning curves for the modular architecture on the "what" and "where" tasks using random and blocked training.

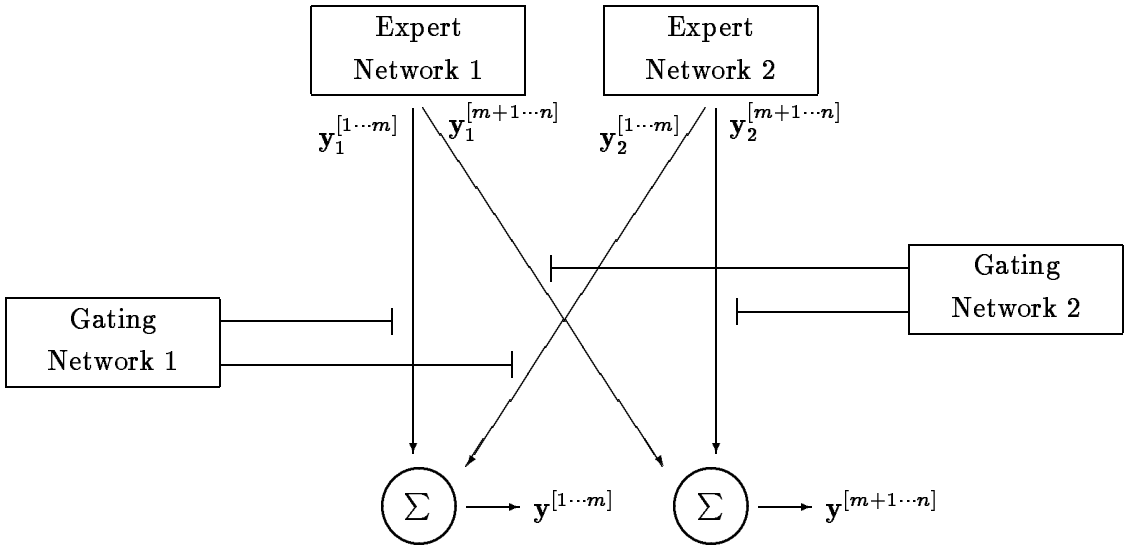

Figure 7: A modular architecture with multiple gating networks ($\mathbf{y}_1^{[1\cdots m]}$, denotes the vector whose components are the first m components of expert network 1's output vector, and the other expressions similarly denote subvectors).

Consider the architecture illustrated in Figure 7. 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 the same expert network to both functions, or they may allocate different expert networks to each function.[4]

[4]Assuming that the output units of the gating network produce binary outputs, the training of each expert network is identical to the case where an external teacher provides desired values for some output

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 Section 2 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 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 net-

---

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 [28] for a discussion of "don't care" conditions and spatial crosstalk).

works (Equation 5) 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 7 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 7 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 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 $5\,X\,5$ 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 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

Expert Networks	Expert Networks' Input	Gating Networks	Gating Networks' Input
$25 \rightarrow 36 \rightarrow 18$	$5\,X\,5$ matrix	$0 \rightarrow 2$	none
$25 \rightarrow 18$	$5\,X\,5$ matrix	$0 \rightarrow 2$	none

Table 3: The modular architecture studied in the spatial crosstalk experiment.

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 2 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 outputs corresponding to the winning expert network increase toward one, and the outputs corresponding to the losing expert networks decrease toward zero. If the current performance does not show significant improvement, then each 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 ap-

proached 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.[5] 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 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 7) 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 7, contains the factor λ_{NT} which is non–zero only when the architecture's performance has not significantly improved.[6] 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

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

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

training.[7] 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 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. [46]. 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. [46]. Our major goal has been

[7]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.

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.

5 Task Decomposition and Network Architectures

The previous section reported the ability of the modular architecture to perform task decomposition on the "what" and "where" vision tasks. In this section, we consider some domain–independent issues concerning task decomposition and discuss 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. [11], le Cun [34], Poggio and Girosi [42]). 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. If there are reasons to prefer one decomposition over another, then it is necessary to use domain knowledge in or-

der 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.

The design of an appropriate architecture, whether for generalization or task decomposition, may use 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 incorporating prior knowledge to bias the nature of the decompositions to be formed. 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 (Jacobs [26], Hampshire and Waibel [21]).

6 Conclusions

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 [18]). 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 paper 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.

7 Appendix

This appendix provides details about the simulations that are not included in the main body of the paper.

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.

	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

Table 4: Parameter values used in temporal crosstalk experiments.

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.

Parameter values

- Temporal crosstalk experiments—For the single networks trained with the backpropagation algorithm, we used the step sizes and momentum that roughly give the best performance. These values are listed in Table 4.

  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 4. 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 5. These values are listed in Table 4.

- 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.

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

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

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

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

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

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

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

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

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

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

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

[21] 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.

[22] 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.

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

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

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

[26] Jacobs, R.A. (in preparation) *Task decomposition through competition in a modular connectionist architecture.* PhD thesis, University of Massachusetts, Amherst, MA.

[27] 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.

[28] 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.

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

[30] 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.

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

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

[33] 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.

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

[35] Maxwell, T., Giles, C.L., Lee, Y.C., & Chen, H.H. (1986) Nonlinear dynamics of artificial neural systems. *Neural Networks for Computing, AIP Conference Proceedings 151.*

[36] 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.

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

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

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

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

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

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

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

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

[53] 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.

[54] 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.

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