The LEABRA Model of Neural Interactions and Learning in the Neocortex

Randall C. O'Reilly

Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
oreilly+@cmu.edu

August 21, 1996

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy

Abstract

There is evidence that the specialized neural processing systems in the neocortex, which are responsible for much of human cognition, arise from the action of a relatively general-purpose learning mechanism. I propose that such a neocortical learning mechanism can be best understood as the combination of error-driven and self-organizing (Hebbian associative) learning. This model of neocortical learning, called LEABRA (local, error-driven and associative, biologically realistic algorithm), is computationally powerful, has important implications for psychological models, and is biologically feasible. The thesis begins with an evaluation of the strengths and limitations of current neural network learning algorithms as models of a neocortical learning mechanism according to psychological, biological, and computational criteria. I argue that error-driven (e.g., backpropagation) learning is a reasonable computational and psychological model, but it is biologically implausible. I show that backpropagation can be implemented in a biologically plausible fashion by using interactive (bi-directional, recurrent) activation flow, which is known to exist in the neocortex, and has been important for accounting for psychological data. However, the interactivity required for biological and psychological plausibility significantly impairs the ability to respond systematically to novel stimuli, making it still a bad psychological model (e.g., for nonword reading). I propose that the neocortex solves this problem by using inhibitory activity regulation and Hebbian associative learning, the computational properties of which have been explored in the context of self-organizing learning models. I show that by introducing these properties into an interactive (biologically plausible) error-driven network, one obtains a model of neocortical learning that: 1) provides a clear computational role for a number of biological features of the neocortex; 2) behaves systematically on novel stimuli, and exhibits transfer to novel tasks; 3) learns rapidly in networks with many hidden layers; 4) provides flexible access to learned knowledge; 5) shows promise in accounting for psychological phenomena such as the U-shaped curve in over-regularization of the past-tense inflection; 6) has a number of other nice properties.

Contents

Acknowledgments

This work has benefitted from the suggestions and comments of a number of people, whom I would like to thank for their time and thoughtfulness: First of all, my advisor and the chair of my committee, Jay McClelland, who was always informative, insightfully critical, supportive, and whose trust in my ability to pursue this topic was very important (especially in the early days!). My partner Yuko Munakata, who provided constant discussion, insight, criticism, support, and love! The outside member of my committee, Geoffrey Hinton, for providing so much of the framework on which these ideas have been built. The other members of my committee, Jon Cohen and John Anderson, who took the time to read at least some of this thing, and provided important perspectives in evaluating this work. Mark Johnson, who was a committee member for a couple of years while he was at CMU. Present and former members of the PDP research group at CMU, especially Dave Plaut, Rich Zemel, Javier Movellan, Jim Hoeffner, Mark Chappell, Craig Stark, Todd Braver, and Rick Gilmore, who sat through several talks on this work, and had many useful comments. Martha Alibali, Marlene Behrmann, Ken Koedinger, and Ken Kotovsky provided useful feedback based on the job talk version on this work. Javier Movellan and Peter Dayan were especially important for the development and presentation of the GeneRec algorithm, and G. Bard Ermentrout got me thinking about second-order integration methods. Larry Wasserman provided some early guidance on Bayesian hypothesis testing. Chad Dawson (and Jay) helped make the PDP++ software a reality, which in turn provided a very useful platform in which to implement and test the LEABRA algorithm (and all the others which are discussed in this thesis). Deanna Barch, Alex Hawkinson, and Shaun Vecera (along with Craig Stark, Jon Cohen, and Todd Braver) were bold (foolish?) enough to attempt some early applications of LEABRA, from which much was learned (at least by me..). Barb Dorney provided much appreciated assistance over the years. Priti Shah was my office mate and friend through all of this. Chloe and Neko patiently sat through several practice talks at home ;)

I would like to acknowledge the financial support from an ONR graduate fellowship, the NPC program funded by an NSF training grant, an NSF equipment grant awarded to Jay McClelland for the computational resources that were used in this research, and a Human Frontiers grant (PI Edmund Rolls) for Hippocampal research that provided an important precursor to this work.

Finally, I would like to thank my family, Richard O'Reilly, Anna O'Reilly, Joan Sweeney, and Katie O'Reilly for their love and support over the years. I would also like to thank my "in-law" family, Mutsuko, Toshinori, Junko, and Naoko Munakata, and Dave Marr, for their love and support.

Chapter 1

Introduction

The neocortex, that wrinkled sheet of neurons upon which all of our thoughts rest, plays the central role in the majority of our higher cognitive functions, from perceptual processing to abstract thinking, language processing, planning, and problem solving. There are three intriguing facts about the neurobiology of the neocortex which form the inspiration for the work presented in this thesis:

1. Despite some level of regional variation, all areas of the neocortex share basic patterns of connectivity and neuron types.[1] (Creutzfeldt, 1977; Szentágothai, 1978; Shepherd, 1988; Douglas, Martin, & Whitteridge, 1989).

2. The development of representations in the neocortex relies in large part on an ontogenetic learning process based on the activity signals of cortical neurons (Hubel & Wiesel, 1965, 1970; Hubel, Wiesel, & LeVay, 1977; Stryker & Harris, 1986). As a dramatic example, it is possible to switch the auditory and visual inputs to the areas of neocortex that usually process this information, and end up with a "visual" area that processes auditory information (Sur, Garraghty, & Roe, 1988).

3. The mechanisms of synaptic modification in all areas of the neocortex appear to be the same, and are based on NMDA receptor dependent long term potentiation (LTP) and depression (LTD) (Artola & Singer, 1993; Linden, 1994).

Thus, it seems that an activity-based learning mechanism, operating within a largely domain-general neural processing framework, could be responsible for creating the powerful representations and processing systems that give rise to much of human cognition. Further, given that the adult cortex exhibits similar forms of activity-dependent plasticity (see Kaas, 1991; Merzenich & Sameshima, 1993, for reviews), it is possible that similar principles are operating in both early (developmental) and

[1] While there are cytoarchitectonically detectable differences in cortical areas, which give rise to the divisions due to Brodman and others, Creutzfeldt (1977) and Shepherd (1988) argue that these reflect differences in the size of afferent and efferent pathways, and not intrinsic circuitry.

mature learning. While there are undoubtedly important specializations in different cortical areas, it might be useful to consider these as different parameterizations of a common underlying mechanism.

Starting with this idea of a general neocortical learning mechanism as a hypothesis, this thesis focuses on the following basic questions regarding the functional and biological properties of such a mechanism:

- What are the signals that drive learning in the cortex? Are error signals or reinforcement signals necessary, or can learning operate purely on "input" signals alone?
- Are there computational advantages associated with using one or more of these different kinds of learning signals?
- Do biological and/or behavioral data provide clear constraints on the nature of learning in the cortex?
- How might the biology of the cortex implement learning based on computationally advantageous learning signals?

Being able to answer these kinds of questions about the nature of a neocortical learning system should lead to a deeper understanding of the nature of the representations and processing that underlie human cognition, in addition to providing support for the idea that the neocortex employs such a general-purpose learning mechanism. Indeed, the potential centrality and generality of the neocortical learning mechanism can be compared with the central role of DNA in biological systems — in both cases a relatively simple mechanism lies at the foundation of a very complex system. However, the cortical learning mechanism is probably much more complex than the structure of DNA, and the connection between its properties and those of complex cognitive phenomena may be difficult to establish. Nevertheless, it seems likely that the understanding of this mechanism will play an important role in the future of psychological investigation.

There are a number of different research strategies that could be, and are being, applied to answering questions like those posed above. Perhaps the most direct approach would be to develop a complete understanding of the biological mechanisms underlying cortical learning. However, given the tremendous complexity of the cortical neurobiology and the huge space of possible variables affecting learning, this approach might be impossible without focusing on particularly useful forms of learning, identified on the basis of their functional properties, that the cortex could plausibly be employing. Thus, the goal of this thesis is to compare several computationally powerful learning algorithms that could plausibly be implemented in the cortical biology, with the objective of providing functional constraints on the general form of cortical learning.

The functional criteria for evaluating these different algorithms are based on general properties that have been identified by existing research as important for modeling psychological phenomena

and learning to solve complex tasks. The ability to exhibit these properties while performing psychologically relevant (but somewhat abstracted) tasks is the functional criterion required of the cortical learning model developed in this thesis. The critical functional properties are as follows:

Interactive (bi-directional, recurrent) processing: Allows both bottom-up and top-down constraints to inform problem solutions (McClelland & Rumelhart, 1981; Plaut & Shallice, 1993), and allows knowledge to be accessed flexibly by enabling different subsets of knowledge at different times to constrain processing. Bidirectional connectivity is a prominent feature of the neocortical circuitry, and is physiologically relevant (Douglas, Koch, Mahowald, Martin, & Suarez, 1995; Douglas & Martin, 1990). Also, this thesis shows that interactivity is important for performing error-driven learning in a biologically plausible fashion.

Systematic responses to novel stimuli: Distributed representations developed over learning provide an effective basis for *generalization*, or treating novel items systematically according to regularities present in training items. For example, this has recently been demonstrated for the case of pronouncing nonwords by Plaut, McClelland, Seidenberg, and Patterson (1996).

Hierarchical, multi-layered representations: Provide efficient solutions to complex, difficult problems by enabling them to be re-represented over multiple stages of processing in ways that make them easier to solve.

Despite the fact that these properties of neural network algorithms are widely regarded as important, it is shown in this thesis that no single existing neural network learning model meets all of these basic functional criteria. Thus, it appears that there is room for improvement over existing models.

In addition to satisfying the functional criteria, it is important that a model of learning in the neocortex provide a computational role for prominent aspects of the cortical neurobiology, and that it not violate basic biological feasibility constraints. Some of the most basic biological properties of the neocortex that should have important functional consequences, and have been discussed in existing computational models, include:

- Pervasive inhibitory circuitry implemented by inhibitory interneurons.
- Prominent recurrent excitatory feedback connectivity between principal (pyramidal) neurons.
- Dominant role of positive-only neural signals in communicating between cortical pyramidal neurons.
- Observed associative character of synaptic long term potentiation (LTP) and depression (LTD).

The biological feasibility constraints at their most basic amount to the use of signals for controlling learning that are locally available to neurons, and are of a form that neurons are likely to be able

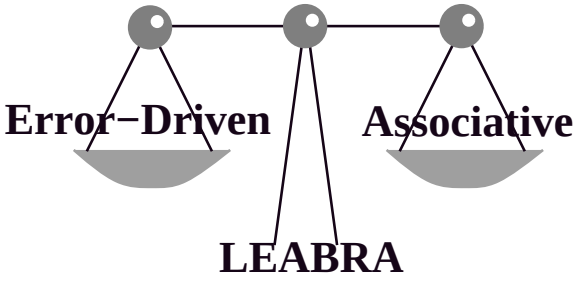

Figure 1.1: A graphic representation of the central idea behind the LEABRA algorithm and its name: a balance between error-driven and self-organizing, associative learning.

to compute. As with the functional criteria, existing algorithms fail to meet all of these biological criteria. Interestingly, this is especially true of the algorithms which come closest to satisfying the above functional criteria.

An important claim of this research is that existing computational learning models fail to provide a satisfactory model of cortical learning according to the basic functional and biological criteria given above. However, different aspects of the above criteria can be satisfied by the two predominant types of existing models. In particular, *error-driven feedforward neural networks* (e.g., *backpropagation*) satisfy the largest portion of the functional criteria, while *self-organizing, Hebbian associative neural networks* satisfy the largest portion of the biological constraints. Still, there are some aspects of both the functional and biological constraints which are satisfied by neither of these two types of algorithms. Nevertheless, it is possible that an algorithm which *combined* both error-driven and self-organizing learning would not only retain their respective advantages, but also somehow satisfy the remaining functional and biological criteria. This would require a way of avoiding the biological feasibility problems associated with error-driven learning, enabling a combined algorithm to satisfy all of the biological criteria.

The central hypothesis explored in this work is that a combined error-driven and self-organizing learning algorithm satisfies all of the basic functional and biological constraints for a model of cortical learning, and therefore represents the best existing model thereof. The overall name for this model of cortical learning is *LEABRA*[2], for "Local, Error-driven and Associative[3], Biologically Realistic Algorithm," which is pronounced like the astrological sign "libra", suggesting a balance between different constraints (see Figure 1.1 for a graphic representation of this idea). It is important to note that, while this name emphasizes the Hebbian associative aspect of self-organizing learning, it is actually both this aspect and the use of constraints on unit activations that are central to the LEABRA algorithm.

There are three main parts of the thesis: **1)** The derivation and testing of a biologically feasible form of error-driven learning (Chapter 2); **2)** The development of theoretical principles behind, and

[2]This name was suggested by Yuko Munakata.
[3]The term *associative* is used here as a synonym for Hebbian associative and self-organizing learning.

an implementation of, the Hebbian associative learning and activity constraints which are used in conjunction with the biologically feasible form of error-driven learning in LEABRA (Chapters 3 and 4); **3)** Testing the theoretical claims regarding the performance of LEABRA compared to several standard algorithms through a number of simulations on a variety of tasks (Chapters 5 through 7). These aspects of the thesis are summarized briefly below, followed by a more extended introduction to the central theoretical claims regarding the functional and biological properties of the LEABRA algorithm.

Chapter 2 presents a biologically plausible form of error-driven learning called *GeneRec* that implements a form of error backpropagation based on some insights derived from the recirculation algorithm of Hinton and McClelland (1988). It is shown that recurrent activation propagation can be used to communicate useful error gradients which approximate those computed by explicit error backpropagation under certain conditions, and that all currently-known error-driven algorithms that use local activation signals to drive learning can be derived from the GeneRec algorithm. The importance of interactive (bidirectional, recurrent) connectivity for communicating error signals in GeneRec is consistent with the extensive level of reciprocal connectivity observed in the cortex (Douglas & Martin, 1990).

Chapter 3 presents a theoretical framework for understanding how self-organizing learning can be viewed as enforcing a set of biases or *a priori* constraints on the way an otherwise purely error-driven system learns. These biases encourage the formation of representations of a generally useful form for a wide range of psychologically relevant tasks according to three representational principles: 1) *Entropy reduction*, which encourages the development of reduced representations of the environment that categorize and filter the input signal; 2) *Information preservation*, which encourages representations to retain informative distinctions in the environment[4]; 3) *Standard representational form*, which is a way of eliminating excess degrees of freedom in the formation of representations. Learning and generalization abilities of networks are improved by biasing the representations in this way, which is an important part of how the combined self-organizing and error-driven learning in LEABRA can satisfy the additional functional criteria that cause problems for purely error-driven algorithms. Finally, this chapter presents a set of implementational principles that influence the way in which these representational principles should be implemented in a neural network.

Chapter 4 presents a neural network algorithm that implements a version of the three representational principles from Chapter 3 according to the aforementioned implementational principles. The important features of this algorithm include: 1) A mechanism that imposes constraints on unit activities called *ReBel*, that encourages the development of entropy reducing representations, and is intended to capture the effects of the inhibitory interneurons in the cortex; 2) Constraints on the signs of weights and activations that constitute a standard representational form, and are consistent with those of cortical neurons; 3) An associative learning algorithm called *MaxIn* which dynamically balances

[4] Note that principles 1 and 2 are in conflict, and the learning algorithm must strike a balance between them.

the entropy reduction and information preservation biases, and is consistent with known properties of associative LTP and LTD in the neocortex. Thus, the LEABRA model consists of an interactive (recurrent) network using the ReBel activation function and a combination of the MaxIn associative and GeneRec error-driven learning rules operating at every connection (synapse) in the network. It is important to emphasize that LEABRA provides a homogeneous, generalized model of learning, not a "hybrid" where one part of the network does error-driven learning and another does self-organizing learning.

Finally, in order to have a combination of error-driven and associative learning pressures operating throughout the network, without restricting the source of error signals to be the output layer only, an *auto-encoder* version of LEABRA is developed. This is simply a LEABRA network that, in addition to solving the desired input-output mapping task, is also trained to maintain the entire input-output activity pattern without any external signals. This ensures that the internal representations have encoded the information present in the input-output pattern, and provides a source of error signals throughout the network as it learns to do so. Simulation results show that this auto-encoder version of LEABRA out-performs the standard version of LEABRA (and all other algorithms tested) on almost all of the tasks studied.

Chapter 5 presents an overview of the simulation models and tasks used to compare the LEABRA model to existing algorithms. Further, this chapter provides a basic demonstration of the differences between LEABRA and the other algorithms on a standard handwritten digit recognition task. **Chapter 6** explores the issue of generalization in systematic or regular domains in greater detail, illustrating the specific problems with interactive error-driven networks. **Chapter 7** explores tasks which require the development of representations over multiple hidden layers, which clearly reveal the advantages of the stronger biases in LEABRA. Finally, **Chapter 8** provides a discussion of these results and of preliminary findings in models of psychological tasks, which indicate some of the future directions of this research.

The remaining sections of this chapter provide a more detailed introduction to the key ideas behind the LEABRA model and its psychological and biological consequences.

Brief Summary of Existing Learning Algorithms

The LEABRA model is based in part on a combination of ideas from existing error-driven and associative (self-organizing) learning algorithms. This section motivates the use of these two forms of learning based on their individual merits, and discusses other possible types of learning signals that the neocortex could plausibly use. In broad summary, there are three categories of neural network learning algorithms, aligned along the dimension of the nature of the signal that drives learning:

- *Error-driven* algorithms, which learn on the basis of the difference between produced outputs and *target* output signals.

- *Reinforcement* algorithms, which learn on the basis of positive and/or negative reward signals in response to behavior.

- *Self-organizing* algorithms, which learn by applying general organizing principles to the input signals without any feedback with respect to performance. These learning rules are typically of a Hebbian associative nature.

While there is evidence that suggests the existence of both reinforcement and Hebbian associative learning in the brain, the same can not be said for error-driven learning (e.g., Crick, 1989). There are two problems typically cited with error-driven learning: the nature and origin of the target signals, and the nature of the error propagation mechanism. The first problem can be addressed by viewing error-driven learning as the process of making implicit predictions about future outcomes, where the future outcome itself serves as a target signal that can drive learning (McClelland, 1994). The second problem can be avoided by using interactive activation propagation to communicate error signals as is done in the biologically plausible GeneRec learning algorithm. Both of these issues are discussed at length in Chapter 2 in the context of the derivation of the GeneRec algorithm. In summary, it does not seem possible to rule out any of the three general forms of learning on the basis of biological plausibility at this point.

While error-driven learning is perhaps the most tenuous from a biological perspective, it is the most computationally powerful of the three algorithms. While different algorithms may perform better on some tasks and worse on others, it is generally true that error-driven algorithms are much better at learning to perform a given task, simply because they have better information available about what they are doing wrong. Even in the simple case of a two-layer pattern-associator network, it is well known that the error-driven delta rule can learn a wider range of input-output mappings than a Hebbian associative algorithm (McClelland & Rumelhart, 1988). While reinforcement algorithms involve the use of error feedback signals, they are less powerful because the error signal carries only one bit of information, as opposed to a more specific signal regarding details about how the output differed from the target.[5]

It is the computational power of error-driven learning which suggests that it must be used in the neocortex, given the complexity of the tasks which are solved by cortical processing. Further, error-driven learning has been used in a large number of psychological models to shape representations according to the contingencies of experience in ways that appear to match the effects of experience on human performance. Indeed, in terms of the functional criteria listed above, it is shown in this thesis that feedforward networks using a standard error-driven (backpropagation) learning algorithm are capable of using graded and distributed representations to solve tasks and generalize well to novel cases.

[5] However, note that reinforcement learning algorithms also incorporate additional mechanisms not often used in standard error-driven learning algorithms that can lead to good performance on some tasks despite the impoverished nature of the feedback signal.

However, interactive (recurrent) versions of these networks do not generalize well at all. Further, these error-driven algorithms do not typically perform well in networks with multiple hidden layers, showing a deficiency in the ability to form effective hierarchical or multi-layered representations. While these deficiencies of purely error-driven networks are important, they are not resolved by using purely self-organizing or purely reinforcement-based learning. Indeed, to the extent that reinforcement learning can be viewed as a special case of error-driven learning, it can be subsumed in the discussion of error-driven learning in the cortex. However, since there appear to be specialized non-cortical neural substrates mediating reinforcement learning in the context of conditioning tasks (e.g., the amygdala, septum, inferior olivary nucleus, basal ganglia, cerebellum, etc.), these other structures may be more responsible for reinforcement learning, while the cortex itself is primarily engaged in error-driven learning.

While self-organizing algorithms are not capable in general of learning to solve arbitrary tasks, they have many attractive characteristics from a biological perspective. Self-organizing algorithms typically employ lateral inhibition and Hebbian associative learning — properties that are known to exist in the anatomy and physiology of the neocortex. Further, it is the self-organizing algorithms which have been most closely tied to biologically measurable computation (e.g., the development of structured representations in the low-level visual system, von der Malsburg, 1973; Linsker, 1988; Miller, Keller, & Stryker, 1989). Thus, one way of viewing the LEABRA model is as a means of testing the computational consequences of the biological properties typically incorporated in self-organizing algorithms on tasks usually reserved for error-driven learning because of their difficulty.

While the LEABRA model attempts to provide a unified view of cortical learning by combining both error-driven and associative (self-organizing) learning in a single algorithm, other ways of combining these two forms of learning have been proposed. One commonly-held view is that "low-level" sensory processing is more like a self-organizing network, and "higher-level" cognitive processing is more like an error-driven network. However, this kind of hybrid approach would require that different learning rules operate in different areas of neocortex. This runs counter to the view that the different cortical areas have more similarities than differences (Creutzfeldt, 1977; Szentágothai, 1978). Further, as described below, there is considerable synergy between error driven and associative learning in LEABRA, so that combining them together makes both biological and computational sense.

Functional Consequences of Error-Driven and Associative Learning

This section provides a more detailed theoretical introduction to the reasons why the combination of error-driven and self-organizing learning in LEABRA should interact in a positive way. Before doing so, it should be noted that if instead there was little interaction between the error-driven and associative learning in LEABRA, this approach would still result in a single neural network formalism which exhibits properties found separately in various current algorithms. This would be useful

to demonstrate in principle, at least to answer the "bag of tricks" kinds of criticisms leveled against neural network models, which find fault with the fact that different (and possibly contradictory) computational principles are used to model different psychological phenomena. Indeed, it is the use of learning algorithms to instantiate a coherent set of principles thought to be central for explaining cognitive phenomena (e.g., McClelland, 1993) which gives them an important role in psychology.

However, the arguments presented in Chapter 3 and the results of LEABRA simulations show that error-driven and associative learning do interact when combined. The following sections outline the specific ways in which the *a priori* biases implemented in LEABRA help to facilitate otherwise purely error-driven learning.

Generalization in Interactive Networks

The standard treatment of the effects of stronger biasing or constraints in neural networks and statistical estimation tends to emphasize the role of constraints or *regularizers* in improving the generalization capabilities of the network (e.g., Weigand, Rumelhart, & Huberman, 1991; Abu-Mostafa, 1989; Geman, Bienenstock, & Doursat, 1992). The idea is that a more constrained or appropriately biased network will not be able to learn as much about the noise in a training set, and will thus avoid the over-fitting problem which leads to poorer performance on testing items drawn from the same underlying distribution as the training items. However, it is not clear how central this aspect of generalization is for thinking about cortical learning, since the human learning process takes place over a long enough time-frame to reduce the aggregate influence of noise to a minimal level. Nevertheless, the tendency to overfit data might be a useful indicator of other important aspects of the learning properties of different algorithms.

There is another way of thinking about generalization which is much more relevant to cognition. Generalization can be viewed as the byproduct of a system which uses representations that capture the *structure* of a given environment. For a simple example, once you know how to count, you can generalize from the sequence $1, 2, 3 \ldots$ to produce further instances in the sequence for as long as you have the patience to do so. Thus, the structure of ordinality and its systematic mapping onto digits enables one to generatively produce and understand the domain of numbers. Clearly, this way of thinking about generalization is central to understanding how humans can exhibit rule-like, generative, and systematic behavior.

Neural network models have been mistakenly characterized as being incapable of exhibiting systematic or generative behavior (e.g., Fodor & Pylyshyn, 1988). There have been many demonstrations that neural network models can learn to perform systematic mappings which appear to conform to the rule-like behavior observed in humans. For example in the domain of language processing, networks learn to perform "regular" or systematic mappings for pronouncing words from orthography, producing the past-tense marking, etc., while at the same time capturing some of the subtlety and irregular behavior found in these domains (Rumelhart & McClelland, 1986; Seidenberg & McClelland, 1989; Plaut et al., 1996). However, despite the inaccuracy of the claim that neural networks can

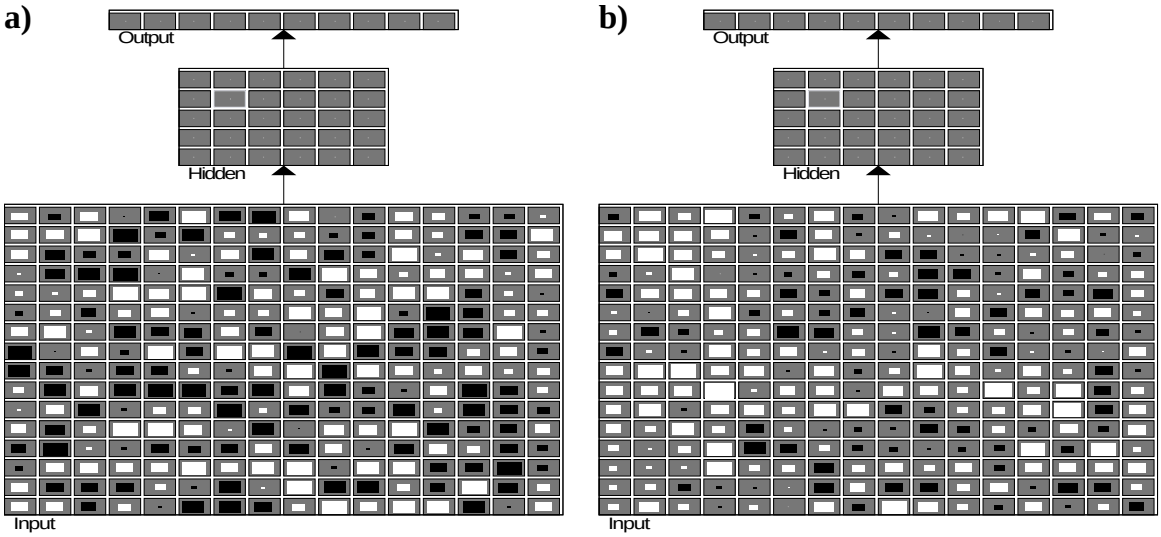

Figure 1.2: Weight values for the same hidden unit in a handwritten digit recognition task before and after training, illustrating the under-constrained nature of feedforward backpropagation networks. Input-to-hidden weights are indicated in the corresponding input unit, with the area indicating magnitude and color indicating sign (white = positive, black = negative)). Given that the inputs consist of visually recognizable digits, these weights, if constrained by the task, should presumably contain recognizable digit-like patterns after training. However, it is not easy to tell which panel shows the random initial weights and which shows them after training, since the trained weights continue to reflect the residue of their random initialization as much as the constraints of the task. In fact a) shows the initial weights, and b) the trained weights.

not exhibit systematic or generative behavior, it is not clear in many cases *why* the network behaves systematically. It is widely known that, except in a few cases where weight decay is particularly useful (e.g., Hinton, 1986; Brousse, 1993) or the task is a very small one like XOR, visual inspection of the weights or activity patterns of units in a backprop network reveals little about how it has solved the task. Thus, while the network as a whole exhibits systematic behavior, it is not immediately clear from examining the behavior or state of its units exactly why this is so. Note that while it is possible to devise analyses that illuminate some aspects of the systematic behavior of networks (see Plaut et al., 1996, for many examples), these typically operate on aggregate behaviors of the system (e.g., cluster analysis of entire activity patterns, correlations of activity patterns, sensitivity of the system to manipulations of input unit activity, etc.). As such, many of these analyses simply amount to different ways of demonstrating the aggregate systematic behavior of the network, and do not specifically say how this arises from the behavior of individual units. This is an example of the kind of problem with neural network models that has led some to complain that they are basically atheoretical (McCloskey, 1991).

One emphasis of this thesis is on a comparative analysis of generalization in different neural network algorithms, with the objective of better understanding its mechanistic basis. The following provides an introduction to the central ideas behind this analysis, and a summary of the results, starting with the feedforward backpropagation algorithm. Backpropagation tends to be “opportunistic” in solving problems, since it only needs to adapt the weights as much as is necessary to produce the

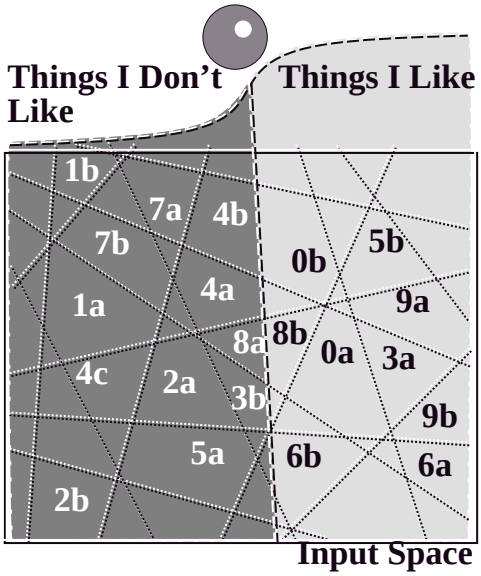

Figure 1.3: Combination of hyperplanes for a digit-recognition problem. The input space, which is actually high-dimensional (of the order of the number of input units), is represented here in two dimensions, with the activity pattern for different input digits occupying different regions of this space. Each hidden unit carves this space in half, along an axis determined by its weights. The combination of a number of such hidden units carves the space into many small regions. Error-driven learning memorizes the correct output for the hidden activity pattern corresponding to each region, and adjusts the locations of the regions so that all inputs within a region share the same output. This does not constrain the weights of an individual unit very much, since responsibility is distributed over many units, and the regions need only be sufficient, not necessary.

correct pattern of outputs, and no further. Thus, if the input-output mapping can be solved in many different ways (as is typically the case), the weights are significantly *under-constrained* by the problem. As a result, the weights tend to reflect the residue of their random initial state as much as the constraints of a given problem, making the units and weights in these networks difficult to interpret (see Figure 1.2 for an example of this phenomenon). Given the enormous numbers of neurons and synapses in the neocortex, it is almost certain that representations there are highly under-constrained.

It is possible to understand how weights can be under-constrained by a given problem, yet still enable the network to solve that problem, in terms of the combination of *hyperplanes* through the input space produced by the hidden units. Figure 1.3 provides an illustration of this in the context of a simple digit-recognition task (see Chapter 5 for actual simulations of this task). Each hidden unit in the network carves the high-dimensional input space in half, responding positively to inputs in one half and negatively to inputs in the other, with the hyperplane dividing these halves determined by its weights. The combination of a number of such hidden units carves the space into many small regions defined by the intersection of their hyperplanes. In order to solve this task, the error-driven learning process essentially memorizes the correct output for the hidden activity pattern corresponding to each region, and adjusts the locations of the regions so that all inputs within a given region have the same output. For most problems, this does not constrain the weights of an individual unit very much, since responsibility for defining the regions is typically distributed over many units, each making a small

contribution to carving up the space. Further, the number and location of the regions need only be sufficient for solving the task, not necessary. Thus, there are undoubtedly many spurious regions left after training.

Despite being under-constrained by the task, feedforward error-driven networks typically generalize surprisingly well. This is probably due to the ability of these networks to treat a novel input with a *graded proportionality* based on the similarity of the novel input to known input patterns. Thus, each hidden unit provides a graded "estimate" of how similar the input pattern is to trained examples along its particular hyperplane, and only if the input pattern should move all the way into its other partition would a hidden unit's activity change significantly (i.e., move to the other side of .5). This same level of robustness exists in the hidden-to-output mapping as well. Further, it is important that many hidden units participate in the representation of each input, so that idiosyncrasies (e.g., residual "randomness") of each hidden unit's representation can be averaged out in the aggregation performed by the output units over all of the hidden units.

To summarize, feedforward error-driven algorithms can typically learn and generalize well with units that are under-constrained by the task. However, despite this apparent success, there are two important problems with purely error-driven learning. First, I show in this thesis that interactive (as opposed to feedforward) error-driven networks do not typically generalize well. As discussed above, interactivity is an important criterion both because of its psychological relevance and its importance for performing biologically-plausible error-driven learning, and it is a prominent feature of neocortical connectivity. Second, I show that under-constrained representations do not typically support useful generalization *across tasks*, which is an important aspect of human generativity — the ability to apply knowledge learned in one task on other tasks. Thus, purely error-driven algorithms do not satisfy all of the functional criteria for a cortical learning model, since they do not simultaneously exhibit good generalization and interactive processing.

The principal reason why interactive error-driven networks do not generalize as well as feedforward networks is that interactivity leads to an increased sensitivity to small differences between input patterns. Thus, instead of treating a novel pattern like the familiar patterns which it closely resembles, an interactive network is more likely to treat it quite differently, resulting in poor generalization performance. Figure 1.4 illustrates this in terms of the *butterfly effect*, which derives its name from the idea that, in the turbulent, interactive dynamics of the weather, a butterfly flapping its wings in China can lead to a hurricane in the Caribbean six months later! To the extent that an interactive network operates in a similarly turbulent activation space, the iterative processing of two similar inputs can diverge quite dramatically over time. However, interactive networks can also exhibit *attractor dynamics*, where similar inputs result in roughly the same final activity state (which can thus be thought of as having a basin of attraction around it). Attractor dynamics should in theory lead to better generalization. The butterfly-effect sensitivity and attractor behavior are two sides of the same coin. Which behavior is expressed in a given network depends on the nature of its weights, which define the relative turbulence or smoothness of the activation space through which interactive

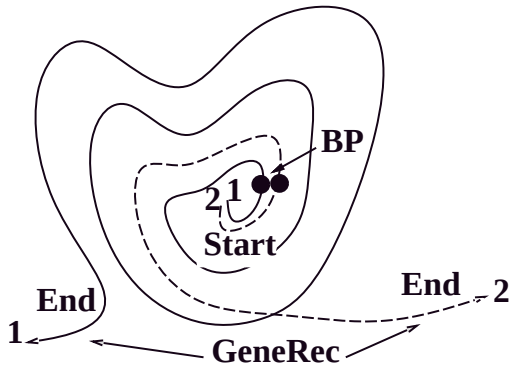

Figure 1.4: Illustration of the *butterfly effect* for interactive networks, where small initial differences are magnified over processing, resulting in large final differences. Shown are 2 initial points, 1 and 2, and their (hypothetical) trajectory through a two-dimensional state space. In a feedforward network (e.g., BP - backpropagation), only one iteration of processing takes place, which does not allow for differences to be magnified significantly. In contrast, an interactive network (e.g., GeneRec, the interactive, biologically plausible form of error-driven learning described in Chapter 2) requires many iterations of updating, allowing differences to be magnified greatly.

settling progresses.

It is because an interactive network is so much more sensitive to noisy weights (which lead to a turbulent activation space) than a feedforward one that the under-constrained nature of the weights in an error-driven network result in worse generalization in this case. This suggests that the way to solve this problem is to eliminate the turbulence in the weights. A number of approaches to this problem have been taken (though typically in a feedforward network), generally involving different ways of imposing further constraints or regularizers on the objective function optimized by learning. Perhaps the most widely used technique is *weight decay*, where weights have a pressure to be zero, so only important weights will remain. Instead of approaching this problem from a purely computational perspective, the hypothesis pursued here is that the neocortex has properties which will provide useful constraints on learning, and that these constraints correspond to the self-organizing (associative) learning component of LEABRA.

There are two properties of LEABRA which solve the problems associated with the under- constrained nature of the weights in a purely error-driven network as described above. The first property is activity competition, which forces hidden units to compete amongst themselves for the ability to represent a given input pattern. This causes individual hidden units to take greater responsibility for representing specific input patterns, and thus causes their weights to more closely correspond to the important distinctions between different inputs. Further, the activity constraints themselves have a *damping* effect on the activation dynamics of the network, reducing its sensitivity. The second property is Hebbian associative learning, which causes a unit's weights to become aligned with the *correlational structure* or *self-structure* of the input patterns. Thus, weights reflect something like the *principal component* or center-of-mass of the inputs for which a given unit is active (Oja, 1982; Linsker, 1988). This results in a much smoother pattern of weights, and thus a smoother activation

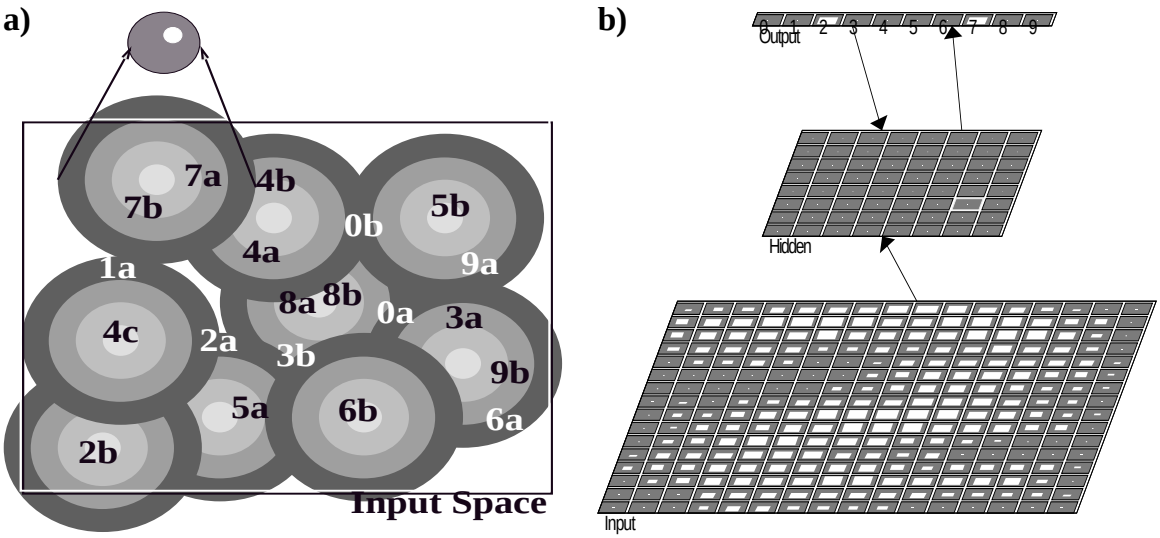

Figure 1.5: Effect of the self-organizing learning in LEABRA on hidden unit weights in the digit recognition task. **a)** Shows a diagram like that of Figure 1.3 for the weights that develop under LEABRA. Instead of each digit being represented by the intersection of many different hyperplanes, individual hidden units develop weights that represent the principal components of the digits, represented by the more constrained receptive fields of the units. This results in "smoother" settling in the network (note the units are not actually radial-basis function units — this is intended to indicate the limited extent of the weights). **b)** Shows a typical set of weights for a LEABRA hidden unit (compare to Figure 1.2), which clearly captures the visual structure of the digits 7 and 2.

space for an interactive network to settle through. The combined impact of these two properties on the representations formed by hidden units in the digit recognition task is shown in Figure 1.5, which can be compared to Figures 1.3 and 1.2. Thus, unlike the purely error-driven network, the weights for LEABRA units clearly reflect the structure of the environment (which is visually salient in this case because the input stimuli are visual representations of digits). The results presented in Chapter 5 show that these weights result in significantly better generalization than an interactive error-driven network, and even somewhat better generalization than a feedforward one. Chapter 6 explores the issue of generalization in greater detail, and provides specific evidence to support the above analysis regarding the nature of generalization in an interactive network.

Finally, it should be noted that the evidence for good generalization in attractor networks reported by Plaut et al. (1996) does not contradict the above analysis because the method used to train their "attractor" network did not actually develop significant weights from the output back to the hidden units, meaning that they were essentially using a feedforward network. Simulations reported in this thesis using the biologically feasible GeneRec error-driven learning algorithm, which requires symmetric bidirectional weights in order to obtain useful error signals, show that a truly interactive network does not typically generalize well.

The Representation of Structure and Cross-Task Generalization

There are important implications of the types of representations formed by LEABRA for the ability of the network to transfer knowledge learned on one task to a novel task (cross-task generalization). Indeed, this type of generalization requires more from the network than simply the ability to exhibit a graded, proportional response to novel stimuli. Instead, it must be able use existing representations of the environment in novel ways. There is reason to believe that the bias in LEABRA towards representing the correlational structure of the environment will be even more useful for cross-task generalization than it is for simple within-task generalization, for the following reasons.

First, correlational structure reflects which features or properties of the environment tend to co-occur, which is an important aspect of the structure of things in the real world, given that co-occurrence is often associated with causal relationships and the meaning of items (e.g., in lexical semantics, co-occurrence of words within a relatively small window of text has been shown to be highly correlated with other measures of semantic relatedness, T. K. Landauer, personal communication). It is important to note that since LEABRA is an interactive network, both input and output patterns contribute equally to defining this correlational structure, which can thus include the correlations between input-output pairs. This input-output correlational structure typically defines the systematicities of the input-output mappings. Thus, in addition to the structure over the different input patterns that is typically thought of in the context of correlational learning, LEABRA units carve up the input space along dimensions that correspond to the systematicities that are predictive of corresponding outputs. Further, the sensitivity to correlational structure exists at all levels in a LEABRA network, so that correlations among higher-level, more abstracted representations are as important as lower-level correlations among items directly in the input and output patterns.

At a more basic level, given that the activity of units in a neural network depends on being activated by other units (which therefore must also be active), it seems that correlational relationships are intrinsically meaningful in the context of neural processing systems, since they determine which units are activated in a given context, and which are not. Thus, the tendency of associative learning to clarify and reinforce the basis of support for a given representation should make them useful for a wide range of tasks that tap the same underlying structure captured by these representations. Finally, it should be noted that the error-driven learning component of LEABRA plays a critical role by constraining the associative learning to pick up on functionally relevant aspects of the correlational structure — those aspects that help to actually solve the mapping problem.

In summary, the LEABRA algorithm has a built-in bias to represent the correlational structure of the environment, at multiple levels of abstraction, and influenced by the "consequences" of an input as well as its own intrinsic structure. This bias should cause the network to represent the important structure of the environment to a much greater extent than a purely error-driven network. It is this ability to represent the important structure of the environment in the first place that allows LEABRA to generalize to novel testing patterns that share this structure. The advantage of this approach over a

purely error-driven network should be most evident when a novel task (as opposed to simply a novel stimulus) is processed in an environment the network has been exposed to previously in a different task context. If the LEABRA network has represented the important structure of the elements in a given environment (i.e., what items/properties tend to "go together"), then it should be able to more easily learn to do different things with these elements compared to a network which has not cleanly represented this structure. This issue is explored further in Chapters 5 and 6, and is important for understanding how human knowledge can be so flexibly used and recombined to solve novel tasks, and may have important implications for the ability to verbally access the contents of internal representations. Finally, as an added benefit, it should be more straightforward to explain network behavior as a direct consequence of its individual units.

Learning to Re-represent in Deep Networks

An apparently difficult problem can become much easier to solve if it is represented in the right way. The psychological literature contains many examples of "insight" problems which require one to re-represent a problem in order to arrive at the correct solution. Similarly, it is clear that computation in the brain depends critically upon many levels of processing which re-represent the input signals in successive stages, with each stage building on the computations performed in the previous one. For example, in perhaps the most well- understood cortical system, the visual system of the monkey, there are at least 5 gross, anatomically separated levels of processing before neurons that fire selectively for object stimuli are found, and it is likely that important sequential computations take place within anatomically defined regions, raising the number of functional layers of computation into the tens (Desimone & Ungerleider, 1989; Van Essen & Maunsell, 1983; Maunsell & Newsome, 1987). If the brain instead tried to go directly from retinal input to invariant object representations, the task would likely be impossible.

Thus, it seems clear that a model of learning in the cortex must be capable of developing over multiple hidden layers successive levels of representations that enable it to solve difficult problems. This is another instance where the benefits of the combination of associative and error-driven learning in LEABRA should be evident. It is widely known that adding hidden layers to a backpropagation network almost never improves performance, despite the potential increase in computational power that they add. While it is true in theory that any kind of input/output mapping can be performed in a single hidden layer, proofs of this rely on the ability of a network with a huge number of hidden units to memorize each training example, which does not allow for the kind of generalization and systematicity that are representative of efficient re-representation solutions in deeper networks.

A classic example of a problem which benefits from multiple hidden layers is the "family trees" problem of Hinton (1986), where the network learns the family relationships for two isomorphic families, and is capable of generalizing to relationship instances it was not trained on. It does this by representing in an intermediate hidden layer the individuals in the family so as to make explicit their functional similarities (i.e., individuals who enter into similar relationships are represented sim-

ilarly). However, it turns out that the deep network used in this example makes it difficult for a purely error-driven network to learn, and, in simulations reported below, learning time actually decreases by using a network with a single hidden layer instead of the original deep network (and generalization is impaired, but not completely eliminated). Finally, this poor depth scaling of error-driven learning is even worse in interactive networks, where training times are twice as long as in the feedforward case, and up to 7 times longer than in LEABRA.

Perhaps a useful analogy for understanding why error-driven learning does not work well in deep networks, especially interactive ones, is that of balancing a stack of poles. It is easier to balance one tall pole than an equivalently tall stack of poles placed one atop the other, because the corrective movements made by moving the base of the pole back and forth have a direct effect on a single pole, while the effects are indirect on poles higher up in a stack. Thus, with a stack of poles, the corrective movements made on the bottom pole have increasingly remote effects on the poles higher up, and the nature of these effects depends on the position of the poles lower down. Similarly, the error signals in a deep network have increasingly indirect and remote effects on layers further down (away from the training signal at the output layer) in the network, and the nature of the effects depends on the representations that have developed in the shallower layers (nearer the output signal). With the increased non-linearity associated with interactive networks, this problem only gets worse.

One way to make the pole-balancing problem easier is to give each pole a little internal gyroscope, so that they each have greater stability and can at least partially balance themselves. The self-organizing learning principles built into LEABRA should provide exactly this kind of self- stabilization, since they lead to the formation of useful representations even in the absence of error signals. At a slightly more abstract level, the LEABRA model is generally more constrained than a purely error-driven learning algorithm, and thus has fewer degrees of freedom to adapt through learning. This can be thought of as limiting the range of motion for each pole, which would also make them easier to balance. The hypothesis that LEABRA will learn better than a purely error-driven algorithm in deep networks is explored in Chapter 7, where LEABRA is shown to be capable of learning significantly faster than these error-driven networks, while exhibiting comparable generalization performance.

The issue of learning in deep networks is potentially important for many current psychological models. For example, the spelling-to-sound model described in Plaut et al. (1996) corrects for the weaknesses of an earlier model (Seidenberg & McClelland, 1989), by representing the input and output in a different, more systematic way, thereby allowing the system to better capture the regularities underlying the reading of regular words, and improving performance on nonword reading.[6] It is this kind of re-representation to capture the systematicities of a given domain that seems to underlie the brain's success in solving difficult computational problems, except that here the re-representation is performed by the researchers, not the model! Thus, as the authors themselves note, a more satisfy-

[6] This is not to say that the Seidenberg and McClelland (1989) representations are more like the surface input/output features used in reading aloud, but the Plaut et al. (1996) representations do emphasize the regularities more than both the earlier representations and the surface features.

ing model would be one that discovers an appropriate sequence of re-representations of stimuli that more closely resemble the raw inputs and outputs that are being modeled. There are many other cases in the literature where important assumptions regarding the input and output representations have been made, yet it remains to be demonstrated that such representations would actually be formed in a "deeper" version of the task by the type of algorithm being used to process those representations.

Learning Interactive Tasks

It has been emphasized above that many of the failures of purely error-driven networks are particularly evident when interactive networks are used. However, none of the tasks which have been used so far depend critically on the use of an interactive network. An example of a task which does require such a network, and illustrates some of the psychological importance of such networks, is a version of the Hinton (1986) family trees problem where any two input cues can be given to retrieve a third piece of information. Thus, one can ask how two people are related, who is the mother of a given person, or whose son is a given person, etc. Clearly, people are able to use multiple different subsets of cues to retrieve information and make inferences, and interactive networks are a natural way of modeling such behavior given that they allow information to propagate in any direction in order to answer a given question.

Despite the close relationship between the interactive and standard feedforward versions of the family trees task, it turns out that I was unable to get a standard recurrent error-driven backpropagation algorithm to learn the interactive version. However, the GeneRec biologically-feasible error-driven learning algorithm, which requires symmetry between reciprocal weights and preserves this via a constraint in the algorithm, is able to learn this task in roughly the same amount of time it takes to learn the standard version. It appears that the symmetry constraint is important for making knowledge learned in solving the problem in one direction useful for solving the problem in other directions, and networks without such a constraint suffer from severe interference from learning in different directions. The LEABRA algorithm, which uses GeneRec for its error-driven learning component, also learned the interactive problem easily, in less than half the time it took GeneRec. This result is important because it shows that while it might be possible to get somewhat better generalization out of the backpropagation version of an interactive network, which does not have a symmetry constraint and thus does not exhibit strong butterfly-effect and attractor dynamics, there is a cost associated with this in not being able to learn interactive tasks. Thus, the only algorithm that provides both good generalization and the ability to learn interactive problems is LEABRA.

Additional Functional Implications of LEABRA

There are a number of potential benefits of combining error-driven and associative learning in addition to the generalization and deep networks issues which constitute the main focus of this thesis. Many of these are related to ideas about LEABRA's performance described above, but they focus on different behavioral implications. For example, the structured nature of the representations devel-

oped by LEABRA could have implications for understanding how people are able to perform analogical reasoning. To the extent that individual representations in the network reflect the systematicities in a given domain, it should then be easier for these representations to be used in other domains which share the same systematic relationships. The cross-task generalization studies described in this thesis provide some evidence along these lines, but they fall short of actually modeling human data or addressing the complexities of analogical reasoning tasks. This remains a topic for future research.

Another case in which the use of associative and error-driven learning might be important is in understanding various developmental phenomena. For example, in Karmiloff-Smith (1992) and work by others, the idea that learning continues beyond the point of competence has been developed. In particular, this post-competence learning is described as reflecting a reorganization of the way that concepts are represented. It seems possible that this can be interpreted as the effect of self-organizing learning that continues to operate even in the absence of the error signals required for error-driven learning. LEABRA provides a natural framework for understanding how representations are affected by both error signals and "internal" forces resulting from the self-organizing learning from the beginning of learning to beyond competency.

In particular, the post-competency representational development in standard error-driven neural network models, which might be defined as the period from when the network is producing outputs within some tolerance of the targets to when the error goes to zero, is typically found to be dominated by the process of memorizing the peculiarities of the training set. In contrast, the human learning data shows that representations become more generalized, more categorical, and more accessible to other domains of knowledge (Karmiloff-Smith, 1992). Interestingly, these properties are consistent with what would be expected from LEABRA after the error signals become less dominant in shaping the representations, at which point the self-organizing learning becomes the dominant force. Some evidence for these aspects of LEABRA are provided the various simulations described later, but much more work remains to be done on this issue.

There is another, perhaps simpler, facet of this post-competency issue, which has to do with explaining behavioral evidence for learning when performance is already perfect. For example, there are many examples in the priming literature which suggest that the mere processing of an item will result in faster subsequent processing. This has been explained by McClelland, McNaughton, and O'Reilly (1995) in terms of small weight changes in the cortical systems that represent and process the given item. While, in the absence of error signals, it is clear that a purely error-driven network would not exhibit this phenomenon, the associative learning aspect of the LEABRA algorithm will cause learning to occur based on the mere activity of neurons. This has been confirmed by Stark (1993) in a model of paired-associate priming, where it was found that a model using LEABRA exhibited the correct priming behavior while a purely error-driven network did not. It should be noted that this associativity property is consistent with the conditions necessary to induce synaptic changes in cortical neurons, as will be discussed below.

Finally, while it is true that error-driven learning is generally superior for solving many problems, there are certain tasks where there are specific computational reasons why associative learning is crucial. Thus, instead of focusing on the advantages of associative learning in the context of error-driven learning, one can ask if there is an advantage to including error-driven learning into an otherwise purely associative learning model. In fact, purely associative or self-organizing models often get stuck in locally optimal solutions, due in part to the positive feedback nature of associative learning — once a given unit comes to represent some part of the input space, this typically gets reinforced by making the associations between the inputs and the unit even stronger. The result is often a unit which represents too much of the input space, making distinctions within this space difficult or impossible. In theory, an error-driven component in the learning algorithm will encourage relevant distinctions (i.e., those distinctions necessary to solve the input-output mapping) to be made, thus improving the reliability of otherwise self-organized learning.

Chapter 8 provides preliminary results for some of these ideas, and presents other functional implications of the LEABRA algorithm.

Biological Aspects of LEABRA

LEABRA is proposed specifically as a model of a biological system, the neocortex. However, it is a fairly abstract model, and, despite the incorporation of several significant aspects of cortical biology, the detailed relationship between its computational properties and those of neurons and networks of neurons has yet to be fully specified. As such, it is reasonable to wonder to what extent it can really be taken seriously as a biological model. The argument in favor of viewing it as such relies on the idea that the neocortex can be specified as a computational device that is somewhat abstracted from its specific biological implementation. However, this general kind of argument, articulated clearly by Marr (1982), has been used to justify all sorts of models that I personally, and many researchers in the field, would be reluctant to describe as being in any way biological. The key test for the biological relevance of a computational model should be *the extent to which the computations performed by the entities in the model can be plausibly related to the underlying biological features*. Thus, short of actually demonstrating that some biological mechanism can really perform the prescribed computation, a model which provides a plausible role for a number of known properties of the biology, while simultaneously performing functions known to be carried out by that same biological system, should be considered a useful model of the biology.

As mentioned above, LEABRA provides a computational role for several prominent features of the neocortical neurobiology, including:

- Pervasive inhibitory circuitry implemented by inhibitory interneurons (Douglas & Martin, 1990): Provides activity regulation that causes units to take more responsibility for representing particular stimuli.

- Prominent interactive excitatory feedback connectivity between principal (pyramidal) neurons (Douglas et al., 1995; Douglas & Martin, 1990): Provides a mechanism for communicating error signals using activation-based mechanisms, which is important for performing error-driven learning in a biologically feasible manner. Also important for flexibly accessing knowledge in the network, and many other psychological effects.

- Observed associative character of synaptic long term potentiation (LTP) and depression (LTD) (Bear & Malenka, 1994; Artola & Singer, 1993; Linden, 1994): Biases units to represent the correlational structure of the environment, which improves generalization (both within and across tasks) and learning in deep networks, among other things.

- Dominant role of positive-only neural signals in communicating between cortical pyramidal neurons (given that the the inhibitory effects of pyramidal neurons on other pyramidal neurons is mediated by interneurons which have a very diffuse and non-modifiable inhibitory effect): This establishes a standard representational form for coding information in the cortex.

Thus, in conjunction with its ability to perform psychologically and computationally important learning that is known to be taking place in the neocortex, the LEABRA algorithm satisfies the stated constraints for being an abstracted biological model.

Further, a detailed mapping of the GeneRec algorithm onto biological mechanisms is proposed in Chapter 2. Indeed, the proposed biological mechanism directly supports the idea that both associative and error-driven factors influence synaptic modification in cortical neurons. In addition, biological properties of cortical neurons and networks provide inspiration for several other functional properties of the LEABRA algorithm. However, at this point, LEABRA remains relatively abstract, providing a level of modeling useful for addressing complicated functional questions. A particularly noticeable omission in LEABRA is a detailed account of the laminar structure of the neocortex, which is simply assumed at this point to provide a useful way of arranging input, "hidden", and output layers in primary sensory corticies. A next step in this research would be to develop a more detailed biological model with realistic cortical neurons which could be used to more thoroughly explore the biological validity of the LEABRA abstraction (see Chapter 8).

Levels of Analysis and Neural Network Models

Neural network models can be pitched at many different levels of analysis, from detailed models where units in the model correspond to neurons in the brain, up to very abstract models which either are not localized with respect to neural circuitry, or where units correspond to large functional regions of the brain. By attempting to incorporate several features of neurons and small networks of neurons into a computational formalism, the present work is assuming a roughly one-to-one mapping between units and neurons. However, the psychological issues being addressed here might be thought of as requiring a more abstract level of modeling. While the relationship between psycholog-

ical and neural levels of analysis is a complicated issue, it is possible to justify the use of fine-grained biological constraints for psychological modeling for the following reasons.

First, it is likely that the neocortex can be described as having a *fractal* character, so that it behaves similarly at different levels of analysis. There are two reasons why this should be the case: 1) It is likely that, in the neocortex, the functional characteristics of long-range connectivity are similar to that of local, short-range connectivity. Pyramidal neurons project excitatory synapses both locally and over larger distances to other brain areas. However, the inhibitory interneurons have primarily local connectivity. Nevertheless, the longer-range excitatory connections synapse both directly on other pyramidal neurons, and on local inhibitory interneurons in that area. Thus, both short and long-range connectivity exhibits a balance between excitation and inhibition, meaning that inter-area interactions might behave according to similar principles as those operating within a given area, or within smaller structures such as columns. 2) The individual effects of neurons on other neurons is likely to be roughly similar to the average effect of a population of neurons on another population, to the extent that both are communicating with signals that resemble average firing rates (and both have similar weights, activity levels, etc). Of course, the population can encode much more information, and is more robust to the effects of noise, but if this is held constant between the two, their interactions will be similar. This is the case in almost all computationally-derived abstract models like backpropagation, and is also true of LEABRA. However, there is considerable debate regarding the important signaling variable in neural firing, and some people believe that average rate is not the relevant one. It is not clear what implications other neural coding schemes would have on this fractal property of the neocortex. In any case, what can be said for certain at this point is that a population of LEABRA units behaves similarly to a single, similarly parameterized LEABRA unit, so that if the brain were constructed completely out of LEABRA units, the scaling issue would not invalidate the importance of the biologically-motivated constraints used in LEABRA for psychological modeling.

This fractal view of the neocortex enables one to make a distinction between the *content* and *processing* of information, which is another way of viewing the scaling issue. Thus, one can claim that a neural network model is performing the same kind of processing as a human in a particular task, but on a very reduced problem that lacks much of the detailed information content of the human equivalent. Thus, the model captures some very reduced aspect of the information processing that a human performs, but not with the same richness of content. Of course, many phenomena can become qualitatively different as they get scaled up or down along this content dimension, but it seems reasonable to allow that some important properties might be relatively scale invariant. In mapping this scaling assumption onto the neocortex, one could plausibly argue that each cortical area could be reduced to handle only a small portion of the content that it actually does (e.g., by the use of a 16x16 pixel retina instead of a 16 million x 16 million pixel retina), but that some important aspects of the essential computation on any piece of that information could be preserved in the reduced model. If several such reduced cortical areas were connected, one could imagine having a useful but simplified model of some reasonably complex psychological phenomena. While many of the actual simulations con-

ducted in this thesis are not intended to map onto any known cortical areas, it is still useful to imagine that a similarly configured chunk of neocortex could perform the task performed by the model.

Chapter 2

Biologically Feasible Error-Driven Learning: GeneRec

Given that the overall goal of LEABRA is to provide a model of learning in the cortex that uses both error-driven and associative learning, it is essential that the error-driven component can be performed in a biologically feasible manner. This chapter presents an algorithm called *generalized recirculation* or *GeneRec* which is capable of doing just that.[1]. Unlike the associative learning and other properties of LEABRA, which are more biologically inspired and can be directly related to known properties of the cortex, the error-driven component is inspired more by computational necessity than by specific properties of the cortex. Nevertheless, as will be discussed in greater length later in this chapter, there are a number of ways in which the GeneRec algorithm makes contact with salient properties of the cortex.

A long-standing objection to the error backpropagation learning algorithm (*BP*) (Rumelhart, Hinton, & Williams, 1986a) is that it is biologically implausible (Crick, 1989; Zipser & Andersen, 1988), principally because it requires error propagation to occur through a mechanism different from activation propagation. This makes the learning appear non-local, since the error terms are not locally available as a result of the propagation of activation through the network. Several remedies for this problem have been suggested, but none are fully satisfactory. One approach involves the use of an additional "error-network" whose job is to send the error signals to the original network via an activation-based mechanism (Zipser & Rumelhart, 1990; Tesauro, 1990), but this merely replaces one kind of non-locality with another, activation-based kind of non-locality (and the problem of maintaining two sets of weights). Another approach uses a global reinforcement signal instead of specific error signals (Mazzoni, Andersen, & Jordan, 1991), but this is not as powerful as standard backpropagation.

[1] Note that this chapter is a copy of a paper entitled "Biologically Plausible Error-driven Learning using Local Activation Differences: The Generalized Recirculation Algorithm" which has been submitted for publication in *Neural Computation.* Thus, it is completely self-contained except for the references, which appear at the end of the thesis

The approach proposed by Hinton and McClelland (1988) is to use bi-directional activation *recirculation* within a single, recurrently connected network (with symmetric weights) to convey error signals. In order to get this to work, they used a somewhat unwieldy four-stage activation update process that only works for auto-encoder networks. This paper presents a generalized version of the recirculation algorithm (*GeneRec*), which overcomes the limitations of the earlier algorithm by using a generic recurrent network with sigmoidal units that can learn arbitrary input/output mappings. The GeneRec algorithm shows how a general form of error backpropagation, which computes essentially the same error derivatives as the Almeida-Pineda (*AP*) algorithm (Almeida, 1987; Pineda, 1987b, 1987a, 1988) for recurrent networks under certain conditions, can be performed in a biologically plausible fashion using only locally available activation variables.

GeneRec uses recurrent activation flow to communicate error signals from the output layer to the hidden layer(s) via symmetric weights. This weight symmetry is an important condition for computing the correct error derivatives. However, the "catch-22" is that GeneRec does not itself preserve the symmetry of the weights, and when it is modified so that it does, it no longer follows the same learning trajectory as AP, even though it is computing essentially the same error gradient. The empirical relationship between the derivatives computed by GeneRec and AP backpropagation is explored in simulations reported in this paper.

The GeneRec algorithm has much in common with the *contrastive Hebbian learning* algorithm (*CHL*, a.k.a. the mean field or deterministic Boltzmann machine (DBM) learning algorithm), which also uses locally available activation variables to perform error-driven learning in recurrently connected networks. This algorithm was derived originally for stochastic networks whose activation states can be described by the Boltzmann distribution (Ackley, Hinton, & Sejnowski, 1985). In this context, CHL amounts to reducing the distance between two probability distributions that arise in two phases of settling in the network. This algorithm has been extended to the deterministic case through the use of approximations or restricted cases of the probabilistic one (Hinton, 1989b; Peterson & Anderson, 1987), and derived without the use of the Boltzmann distribution by using the continuous Hopfield energy function (Movellan, 1990). However, all of these derivations require problematic assumptions or approximations, leading some to conclude that CHL is fundamentally flawed for deterministic networks (Galland, 1993; Galland & Hinton, 1990).

It is shown in this paper that the CHL algorithm can be derived instead as a variant of the GeneRec algorithm, which establishes a more general formal relationship between the BP framework and the deterministic CHL rule than previous attempts (Peterson, 1991; Movellan, 1990; LeCun & Denker, 1991). This relationship means that all known fully general error-driven learning algorithms that use local activation-based variables in deterministic networks can be considered variations of the GeneRec algorithm (and thus, indirectly, of the backpropagation algorithm as well). An important feature of the GeneRec-based derivation of CHL is that the relationship between the learning properties of BP, GeneRec, and CHL can be more clearly understood. Another feature of this derivation is that it is completely general with respect to the activation function used, allowing CHL-like learning

rules to be derived for many different cases.

CHL is equivalent to GeneRec when using a simple approximation to a second-order accurate numerical integration technique known as the *midpoint* or second-order Runge-Kutta method, plus an additional symmetry preservation constraint. The implementation of the midpoint method in GeneRec simply amounts to the incorporation of the sending unit's plus-phase activation state into the error derivative, and as such, it amounts to an on-line (per pattern) integration technique. This method results in faster learning by reducing the amount of interference due to independently computed weight changes for a given pattern. This would explain why CHL networks generally learn faster than otherwise equivalent BP networks (e.g., Peterson & Hartman, 1989; Movellan, 1990). A comparison of optimal learning speeds for all variants of GeneRec and feedforward and AP recurrent backprop on four different problems is reported in this paper. The results of this comparison are consistent with the derived relationship between GeneRec and AP backpropagation, and with the interpretation of CHL as a symmetric, midpoint version of GeneRec, and thus provide empirical support for these theoretical claims.

The finding that CHL did not perform well at all in networks with multiple hidden layers ("deep" networks), reported by Galland (1993), would appear to be problematic for the claim that CHL is performing a fully general form of backpropagation, which can learn in deep networks. However, I was unable to replicate the Galland (1993) failure to learn the "family trees" problem (Hinton, 1986) using CHL. In simulations reported below, I show that by simply increasing the number of hidden units (from 12 to 18), CHL networks can learn the problem with 100% success rate, in a number of epochs on the same order as backpropagation. Thus, the existing simulation evidence seems to support the idea that CHL is performing a form of backpropagation, and not that it is a fundamentally flawed approximation to the Boltzmann machine as has been argued (Galland, 1993; Galland & Hinton, 1990).

Given that the GeneRec family of algorithms encompasses all known ways of performing error-driven learning using locally-available activation variables, it provides a promising framework for thinking about how error-driven learning might be implemented in the brain. To further this goal, an explicit biological mechanism capable of implementing GeneRec-style learning is proposed. This mechanism is consistent with available evidence regarding synaptic modification in neurons in the neocortex and hippocampus, and makes further predictions.

Introduction to Algorithms and Notation

In addition to the original recirculation algorithm (Hinton & McClelland, 1988), the derivation of GeneRec depends on ideas from several standard learning algorithms, including: feedforward error backpropagation (BP) (Rumelhart et al., 1986a) with the cross-entropy error term (Hinton, 1989a); the Almeida-Pineda (AP) algorithm for error backpropagation in a recurrent network (Almeida, 1987; Pineda, 1987b, 1987a, 1988); and the contrastive Hebbian learning algorithm (CHL) used in

Layer (index)	Net Input	Activation
Input (s)	—	s_i = stimulus input
Hidden (h)	$\eta_j = \sum_i w_{ij} s_i$	$h_j = \sigma(\eta_j)$
Output (o)	$\eta_k = \sum_j w_{jk} h_j$	$o_k = \sigma(\eta_k)$

Table 2.1: Variables in a 3 layer backprop network. $\sigma(\eta)$ is the standard sigmoidal activation function $\sigma(\eta) = 1/(1+e^{-\eta})$.

the Boltzmann machine and deterministic variants (Ackley et al., 1985; Hinton, 1989b; Peterson & Anderson, 1987). The notation and equations for these algorithms are summarized in this section, followed by a brief overview of the recirculation algorithm in the next section. This provides the basis for the development of the generalized recirculation algorithm presented in subsequent sections.

Feedforward Error Backpropagation

The notation for a three-layer feedforward backprop network uses the symbols shown in table 2.1. The target values are labeled t_k for output unit k, and the pattern-wise sum is dropped since the subsequent derivations do not depend on it. The cross-entropy error formulation (Hinton, 1989a) is used because it eliminates an activation derivative term in the learning rule which is also not present in the recirculation algorithm.

The cross-entropy error is defined as:

$$E = \sum_k t_k \log\, o_k + (1 - t_k) \log(1 - o_k) \tag{2.1}$$

and the derivative of E with respect to a weight into an output unit is:

$$\begin{aligned} \frac{\partial E}{\partial w_{jk}} &= \frac{dE}{do_k}\frac{do_k}{d\eta_k}\frac{\partial \eta_k}{\partial w_{jk}} \\ &= \left[\frac{t_k}{o_k} - \frac{(1-t_k)}{(1-o_k)}\right] \sigma'(\eta_k) h_j \\ &= (t_k - o_k) h_j \end{aligned} \tag{2.2}$$

where $\sigma'(\eta_k)$ is the first derivative of the sigmoidal activation function with respect to the net input: $o_k(1-o_k)$, which is canceled by the denominator of the error term. In order to train the weights into the hidden units, the impact a given hidden unit has on the error term needs to be determined:

$$\begin{aligned} \frac{\partial E}{\partial h_j} &= \sum_k \frac{dE}{do_k}\frac{do_k}{d\eta_k}\frac{\partial \eta_k}{\partial h_j} \\ &= -\sum_k (t_k - o_k) w_{jk} \end{aligned} \tag{2.3}$$

which can then be used to take the derivative of the error function with respect to the input to hidden

unit weights:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial h_j}\frac{dh_j}{d\eta_j}\frac{\partial \eta_j}{\partial w_{ij}} \\ &= -\sum_k (t_k - o_k) w_{jk} \sigma'(\eta_j) s_i \end{aligned} \tag{2.4}$$

which provides the basis for adapting the weights.

Almeida-Pineda Recurrent Backpropagation

The AP version of backpropagation is essentially the same as the feedforward one described above except that it allows for the network to have recurrent (bidirectional) connectivity. Thus, the network is trained to settle into a stable activation state with the output units in the target state, based on a given input pattern clamped over the input units. This is the version of BP that the GeneRec algorithm, which also uses recurrent connectivity, approximates most closely. The same basic notation and cross-entropy error term as described for feedforward BP are used to describe the AP algorithm, except that the net input terms (η) can now include input from any other unit in the network, not only those in lower layers.

The activation states in AP are updated according to a discrete-time approximation of the following differential equation, which is integrated over time with respect to the net input terms[2]:

$$\frac{d\eta_j}{dt} = -\eta_j + \sum_i w_{ij} \sigma(\eta_i) \tag{2.5}$$

This equation can be iteratively applied until the network settles into a stable equilibrium state (i.e., until the change in activation state goes below a small threshold value), which it will provably do if the weights are symmetric (Hopfield, 1984), and often even if they are not (Galland & Hinton, 1991).

In the same way that the activations are iteratively updated to allow for recurrent activation dynamics, the error propagation in the AP algorithm is also performed iteratively. The iterative error propagation in AP operates on a new variable y_j which represents the current estimate of the derivative of the error with respect to the net input to the unit, $\frac{\partial E}{\partial \eta_j}$. This variable is updated according to:

$$\frac{dy_j}{dt} = -y_j + \sigma'(\eta_j) \sum_k w_{kj} y_k + J_j \tag{2.6}$$

where J_j is the externally "injected" error for output units with target activations. This equation is iteratively applied until the y_j variables settle into an equilibrium state (i.e., until the change in y_j falls below a small threshold value). The weights are then adjusted as in feedforward BP (2.4), with y_j providing the $\frac{\partial E}{\partial \eta_j}$ term.

[2] It is also possible to incrementally update the activations instead of the net inputs, but this limits the ability of units to change their state rapidly, since the largest activation value is 1, while net inputs are not bounded.

Layer	Phase	Net Input	Activation
Input (s)	—	—	s_i = stimulus input
Hidden (h)	-	$\eta_j^- = \sum_i w_{ij} s_i + \sum_k w_{kj} o_k^-$	$h_j^- = \sigma(\eta_j^-)$
	+	$\eta_j^+ = \sum_i w_{ij} s_i + \sum_k w_{kj} o_k^+$	$h_j^+ = \sigma(\eta_j^+)$
Output (o)	-	$\eta_k^- = \sum_j w_{jk} h_j^-$	$o_k^- = \sigma(\eta_k^-)$
	+	—	$o_k^+ = t_k$

Table 2.2: Equilibrium network variables in a three layer network having reciprocal connectivity between the hidden and output layers with symmetric weights ($w_{jk} = w_{kj}$), and phases over the output units such that the target is clamped in the plus phase, and not in the minus phase. $\sigma(\eta)$ is the standard sigmoidal activation function.

Contrastive Hebbian Learning

The contrastive-Hebbian learning algorithm (CHL) used in the stochastic Boltzmann machine and deterministic variants is based on the differences between activation states in two different phases. As in the AP algorithm, the connectivity is recurrent, and activation states (in the deterministic version) can be computed according to (2.5). As will be discussed below, the use of locally-computable activation differences instead of the non-local error backpropagation used in the BP and AP algorithms is more biologically plausible. The GeneRec algorithm, from which the CHL algorithm can be derived as a special case, uses the same notion of activation phases.

The two phases of activation states used in CHL are the *plus* phase states, which result from both input and target being presented to the network, and provide a training signal when compared to the *minus* phase activations, which result from just the input pattern being presented. The equilibrium network variables (i.e., the states after the iterative updating procedure of (2.5) has been applied) for each phase in such a system are labeled as in table 2.2.

The CHL learning rule for deterministic recurrent networks can be expressed in terms of generic activation states a (which can be from any layer in the network) as follows:

$$\frac{1}{\epsilon}\Delta w_{ij} = (a_i^+ a_j^+) - (a_i^- a_j^-) \tag{2.7}$$

where a_i is the sending unit and a_j is the receiving unit. Thus, CHL is simply the difference between the pre and post-synaptic activation coproducts in the plus and minus phases. Each coproduct term is equivalent to the derivative of the energy or "harmony" function for the network with respect to the weight:

$$E = \sum_j \sum_i a_j a_i w_{ij} \tag{2.8}$$

and the intuitive interpretation of this rule is that it decreases the energy or increases the harmony of the plus-phase state and vice-versa for the minus-phase state (see Ackley et al., 1985; Peterson & Anderson, 1987; Hinton, 1989b; Movellan, 1990, for details).

The Recirculation Algorithm

The original Hinton and McClelland (1988) recirculation algorithm is based on the feedforward BP algorithm. The GeneRec algorithm, which is more closely related to the AP recurrent version of backpropagation, borrows two key insights from the recirculation algorithm. These insights provide a means of overcoming the main problem with the standard backprop formulation from a neural plausibility standpoint, which is the manner in which a hidden unit computes its own error contribution. This is shown in (2.3). The problem is that the hidden unit is required to access the computed quantity ($\frac{\partial E}{\partial o_k}$), which depends on variables at the output unit only. This is the crux of the non-locality of error information in backpropagation.

The first key insight that can be extracted from the recirculation algorithm is that (2.3) can be expressed as the difference between two terms, each of which look much like a net-input to the hidden unit:

$$\frac{\partial E}{\partial h_j} = -\left(\sum_k t_k w_{jk} - \sum_k o_k w_{jk}\right) \tag{2.9}$$

Thus, instead of having a separate error-backpropagation phase to communicate error signals, one can think in terms of standard activation propagation occurring via reciprocal (and symmetric) weights that come from the output units to the hidden units. The error contribution for the hidden unit can then be expressed in terms of the difference between two net-input terms. One net-input term is just that which would be received when the output units had the target activations t_k clamped on them, and the other is that which would be received when the outputs have their feed-forward activation values o_k.

In order to take advantage of a net-input based error signal, Hinton and McClelland (1988) used an auto-encoder framework, with two pools of units: *visible* and *hidden*. The visible units play the role of both the input layer and the output layer. Each training pattern, which is its own target, is presented to the visible units, which then project to a set of hidden units, which then feed back to the same visible units. The input from the hidden units changes the state of the visible units, and this new state is then fed through the system again, hence the name *recirculation* (see Figure 2.1). As a result, the visible units have two activation states, equivalent to t_k (or s_i) and o_k, and the hidden units have two activation states, the first of which is a function of $\eta_j^* = \sum_k t_k w_{kj}$, which can be labeled h_j^*, and the second of which corresponds to h_j.

The second key insight from the recirculation algorithm is that instead of computing a difference in net-inputs in (2.9), one can use a difference of activations via the following approximation to (2.4):

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= -(\eta_j^* - \eta_j)\sigma'(\eta_j)t_k \\ &\approx -(h_j^* - h_j)t_k \end{aligned} \tag{2.10}$$

The difference between activation values instead of the net-inputs in (2.10) can be used since Hin-

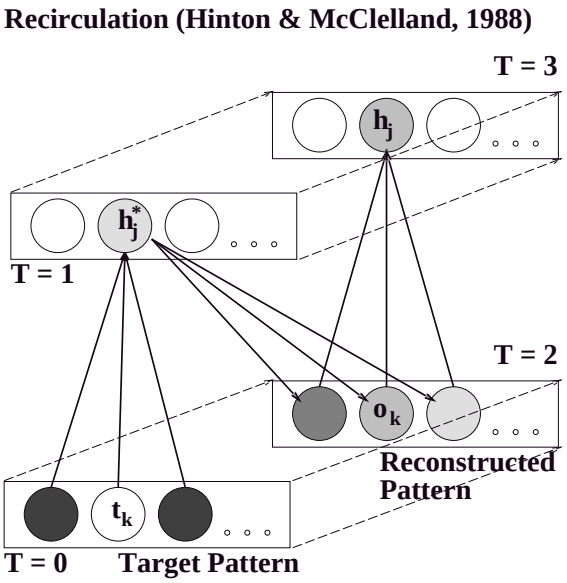

Figure 2.1: The recirculation algorithm, as proposed by Hinton and McClelland (1988). Activation is propagated in four steps ($T = 0 - 3$). $T = 0$ is the target pattern clamped on the visible units. $T = 1$ is the hidden units computing their activation as a function of the target inputs. $T = 2$ is the visible units computing their activations as a function of the hidden unit state at $T = 1$. $T = 3$ is the hidden unit state computed as a function of the reconstructed visible unit pattern.

ton and McClelland (1988) imposed an additional constraint that the difference between the reconstructed and the target visible-unit states (and therefore the difference between η_j^* and η_j) be kept small by using a "regression" function in updating the visible units. This function assigns output state (computed at time $T = 2$) as a weighted average of the target output and the activation computed from the current net input from the hidden units[3]:

$$o_k = \lambda t_k + (1 - \lambda) f(\eta_k) \tag{2.11}$$

Thus, the difference in a hidden-unit's activation values is approximately equivalent to the difference in net-inputs times the slope of the activation function at one of the net-input values ($\sigma'(\eta_j)$), as long as the linear approximation of the activation function given by the slope is reasonably valid. Even if this is not the case, as long as the activation function is monotonic, the error in this approximation will not affect the sign of the resulting error derivative, only the magnitude. Nevertheless, errors in magnitude can lead to errors in sign over the pattern-wise sum.

Since the difference of activations in (2.10) computes the derivative of the activation function implicitly, one can use the resulting learning rule for any reasonable (monotonic) activation function[4].

[3] Note that Hinton and McClelland (1988) used linear output units to avoid the activation function derivative on the output units, whereas cross-entropy is being used here to avoid this derivative. Thus, the function $f(\eta_k)$ in (2.11) will either be linear or a sigmoid, depending on which assumptions are being used.

[4] Note that the output units have to use a sigmoidal function in order for the cross-entropy function to cancel out the derivative.

This can be important for cases where the derivative of the activation function is difficult to compute. Further, the activation variable might be easier to map onto the biological neuron, and it avoids the need for the neuron to compute its activation derivative.

Note that due to the above simplification, the learning rule for the recirculation algorithm (based on (2.10)) is the same for both hidden and output units, and is essentially the delta-rule. This means that locally available activation states of the pre and postsynaptic units can be used to perform error-driven learning, which avoids the need for a biologically troublesome error backpropagation mechanism that is different from the normal propagation of activation through the network.

Phase-Based Learning and Generalized Recirculation

While the recirculation algorithm is more biologically plausible than standard feedforward error backpropagation, it is limited to learning auto-encoder problems. Further, the recirculation activation propagation sequence requires a detailed level of control over the flow of activation through the network and its interaction with learning. However, the critical insights about computing error signals using differences in net input (or activation) terms can be applied to the more general case of a standard three-layer network for learning input to output mappings. This section presents such a *generalized recirculation* or *GeneRec* algorithm, which uses standard recurrent activation dynamics (as used in the AP and CHL algorithms) to communicate error signals instead of the recirculation technique. Instead of using the four stages of activations used in recirculation, GeneRec uses two activation phases as in the CHL algorithm described above. Thus, in terms of activation states, GeneRec is identical to the deterministic CHL algorithm, and the same notation is used to describe it.

The learning rule for GeneRec is simply the application of the two key insights from the recirculation algorithm to the AP recurrent backpropagation algorithm instead of the feedforward BP algorithm, which was the basis of the recirculation algorithm. If the recurrent connections between the hidden and output units are ignored so that the error on the output layer is held constant, it is easy to show that the fixed point solution to the iterative AP error updating equation (2.6) (i.e., where $\frac{dy_j}{dt} = 0$ for hidden unit error y_j) is of the same form as feedforward backpropagation. This means that the same recirculation trick of computing the error signal as a difference of net input (or activation) terms can be used:

$$\begin{aligned} y_j^\infty &= \sigma'(\eta_j) \sum_k w_{kj} y_k^\infty \\ &\approx \sigma'(\eta_j) \left(\sum_k w_{kj} (t_k - o_k) \right) \\ &\approx \sigma'(\eta_j) \left(\sum_k w_{kj} t_k - \sum_k w_{kj} o_k \right) \end{aligned} \tag{2.12}$$

Thus, assuming constant error on the output layer, the equilibrium error gradient computed for hidden units in AP is equivalent to the difference between the GeneRec equilibrium net input states in the plus and minus phases. Note that the minus phase activations in GeneRec are identical to the AP activation states. The difference of activation states can be substituted for net input differences times the derivative of the activation function by the approximation introduced in recirculation, resulting in the following equilibrium unit error gradient:

$$y_j^{\infty} \approx h_j^+ - h_j^- \tag{2.13}$$

Note that while the hidden unit states in GeneRec also reflect the constant net input from the input layer (in addition to the output-layer activations that communicate the necessary error gradient information), this cancels out in the difference computation of (2.13). However, this constant input to the hidden units from the input layer in both phases can play the role of the regression update equation (2.11) in recirculation. To the extent that this input is reasonably large and it biases the hidden units towards one end of the sigmoid or the other, this bias will tend to moderate the differences between h_j^- and h_j^+, making their difference a reasonable approximation to the differences of their respective net inputs times the slope of the activation function.

While the analysis presented above is useful for seeing how GeneRec equilibrium activation states could approximate the equilibrium error gradient computed by AP, the AP algorithm actually performs iterative updating of the error variable (y_j). Thus, it would have to be the case that iterative updating of this single variable is equivalent to the iterative updating of each of the activation states (plus and minus) in GeneRec, and then taking the difference. This relationship can be expressed by writing GeneRec in the AP notation. First, we define two components to the error variable y_j, which are effectively the same as the GeneRec plus and minus phase net inputs (ignoring the net input from the input units, which is subtracted out in the end anyway):

$$\frac{dy_j^+}{dt} = -y_j^+ + \sum_k w_{kj} t_k \tag{2.14}$$

$$\frac{dy_j^-}{dt} = -y_j^- + \sum_k w_{kj} \sigma(\eta_k) \tag{2.15}$$

Then, we approximate the fixed point of y_j with the difference of the fixed points of these two variables:

$$\begin{aligned} y_j^{\infty} &\approx \sigma'(\eta_j)(y_j^{\infty +} - y_j^{\infty -}) \\ &\approx h_j^+ - h_j^- \end{aligned} \tag{2.16}$$

which can be approximated by the subtraction of the GeneRec equilibrium activation states as discussed previously.

Unfortunately, the validity of this approximation is not guaranteed by any proof that this author is aware of. However, there are several properties of the equations that lend some credibility to it. First, the part that is a function of t_k on the output units, y_j^+, is effectively a constant, and the other part, y_j^-, is just the activation updating procedure that both GeneRec and AP have in common. Further, given that the fixed point solutions of the GeneRec and AP equations are the same when recurrent influences are ignored, and the pattern of recurrent influences is given by the same set of weights, it is likely that the additional effects of recurrence will be in the same direction for both GeneRec and AP. However, short of a proof, these arguments require substantiation from simulation results comparing the differences between the error derivatives computed by GeneRec and AP. The results presented later in the paper confirm that GeneRec computes essentially the same error derivatives as AP in a three layer network (as long as the weights are symmetric). This approximation deteriorates only slightly in networks with multiple hidden layers, where the effects of recurrence are considerably greater.

As in the recirculation algorithm, it is important for the above approximation that the weights into the hidden units from the output units have the same values as the corresponding weights that the output units receive from the hidden units. This is the familiar symmetric weight constraint, which is also necessary to prove that a network will settle into a stable equilibrium (Hopfield, 1984). We will revisit this constraint several times during the paper. However, for the time being, we will assume that the weights are symmetric.

Finally, virtually all deterministic recurrent networks including GeneRec suffer from the problem that changes in weights based on gradient information computed on equilibrium activations might not result in the network settling into an activation state with lower error the next time around. This is due to the fact that small weight changes can affect the settling trajectory in unpredictable ways, resulting in an entirely different equilibrium activation state than the one settled into last time. While it is important to keep in mind the possibility of discontinuities in the progression of activation states over learning, there is some basis for optimism on this issue. For example, in his justification of the deterministic version of the Boltzmann machine (DBM) Hinton (1989b) supplies several arguments (which are substantiated by a number of empirical findings) justifying the assumption that small weight changes will generally lead to a contiguous equilibrium state of unit activities in a recurrent network.

To summarize, the learning rule for GeneRec that computes the error backpropagation gradient locally via recurrent activation propagation is the same as that for recirculation, having the form of the delta-rule. It can be stated as follows in terms of a sending unit with activation a_i and a receiving unit with activation a_j:

$$\frac{1}{\epsilon}\Delta w_{ij} = a_i^-(a_j^+ - a_j^-) \tag{2.17}$$

As shown above, this learning rule will compute the same error derivatives as the AP recurrent backpropagation procedure under the following conditions:

- Iterative updating of the error term (y_j) can be approximated by the separate iterative updating of the two activation terms (h_j^+ and h_j^-) and then taking their difference.
- The reciprocal weights are symmetric ($w_{jk} = w_{kj}$).
- Differences in net inputs times the activation function derivative can be approximated by differences in the activation values.

The Relationship Between GeneRec and CHL

The GeneRec learning rule given by (2.17) and the CHL learning rule given by (2.7) are both simple expressions that involve a difference between plus and minus phase activations. This raises the possibility that they could somehow be related to each other. Indeed, as described below, there are two different ways in which GeneRec can be modified that, when combined, yield (2.7).

The GeneRec learning rule can be divided into two parts, one of which represents the derivative of the error with respect to the unit (the difference of that unit's plus and minus activations, $a_j^+ - a_j^-$), and the other which represents the contribution of a particular weight to this error term (the sending unit's activation, a_i^-). It is the phase of this latter term which is the source of the first modification. In standard feedforward or AP recurrent backpropagation, there is only one activation term associated with each unit, which is equivalent to the minus-phase activation in the GeneRec phase-based framework. Thus, the contribution of the sending unit is naturally evaluated with respect to this activation, and that is why a_i^- appears in the GeneRec learning rule.

However, given that GeneRec has another activation term corresponding to the plus-phase state of the units, one might wonder if the derivative of the weight should be evaluated with respect to this activation instead. After all, the plus phase activation value will likely be a more accurate reflection of the eventual contribution of a given unit after other weights in the network are updated. In some sense, this value *anticipates* the weight changes that will lead to having the correct target values activated, and learning based on it might avoid some interference.

On the other hand, the minus phase activation reflects the *actual* contribution of the sending unit to the current error signal, and it seems reasonable that credit assignment should be based on it. Given that there are arguments in favor of both phases, one approach would be to simply use the average of both of them. Doing this results in the following weight update rule:

$$\frac{1}{\epsilon}\Delta w_{ij} = \frac{1}{2}(a_i^- + a_i^+)(a_j^+ - a_j^-) \tag{2.18}$$

This is the first way in which GeneRec needs to be modified to make it equivalent to CHL. As will be discussed in detail below, this modification of GeneRec corresponds to a simple approximation of the *midpoint* or second-order accurate Runge-Kutta integration technique. The consequences of the midpoint method for learning speed will be explored in the simulations reported below.

The second way in which GeneRec needs to be modified concerns the issue of symmetric weights. In order for GeneRec to compute the error gradient via reciprocal weights, these weights need to have the same value (or at least the same relative magnitudes and signs) as the forward-going weights. However, the basic GeneRec learning rule (2.17) does not preserve this symmetry:

$$a_i^-(a_j^+ - a_j^-) \neq a_j^-(a_i^+ - a_i^-) \tag{2.19}$$

While simulations reported below indicate that GeneRec can learn and settle into stable attractors without explicitly preserving the weight symmetry, a symmetry-preserving version of GeneRec would guarantee that the computed error derivatives are always correct.

One straightforward way of ensuring weight symmetry is simply to use the average (or more simply, the sum) of the weight changes that would have been computed for each of the reciprocal weights separately, and apply this same change to both weights. Thus, the symmetric GeneRec learning rule is:

$$\begin{aligned} \frac{1}{\epsilon}\Delta w_{ij} &= a_i^-(a_j^+ - a_j^-) + a_j^-(a_i^+ - a_i^-) \\ &= (a_j^+ a_i^- + a_j^- a_i^+) - 2a_j^- a_i^- \end{aligned} \tag{2.20}$$

Note that using this rule will not result in the weights being updated in the same way as AP backpropagation, even though the error derivatives computed on the hidden units will still be the same. Thus, even the symmetry preserving version of GeneRec is not identical to AP backpropagation. This issue will be explored in the simulations reported below.

If both the midpoint method and symmetry preservation versions of GeneRec are combined, the result is the CHL algorithm:

$$\begin{aligned} \frac{1}{\epsilon}\Delta w_{ij} &= \frac{1}{2}\left[(a_i^+ + a_i^-)(a_j^+ - a_j^-) + (a_j^+ + a_j^-)(a_i^+ - a_i^-)\right] \\ &= (a_i^+ a_j^+) - (a_i^- a_j^-) \end{aligned} \tag{2.21}$$

Note that LeCun and Denker (1991) pointed out that CHL is related to a symmetric version of the delta rule (i.e., GeneRec), but they did not incorporate the midpoint method, and thus were only able to show an approximation that ignored this aspect of the relationship between CHL and GeneRec.

The above derivation of CHL is interesting for several reasons. First, it is based on error backpropagation (via GeneRec), and not some kind of approximation to a stochastic system. This eliminates the problems associated with considering the graded activations of units in a deterministic system to be expected values of some underlying probability distribution. For example, in order to compute the probability of a given activation state, one needs to assume that the units are statistically independent (see Hinton, 1989b; Peterson & Anderson, 1987). While Movellan (1990) showed that CHL can be derived independent of the Boltzmann distribution and the concomitant mean-field assumptions, his derivation does not apply when there are hidden units in the network.

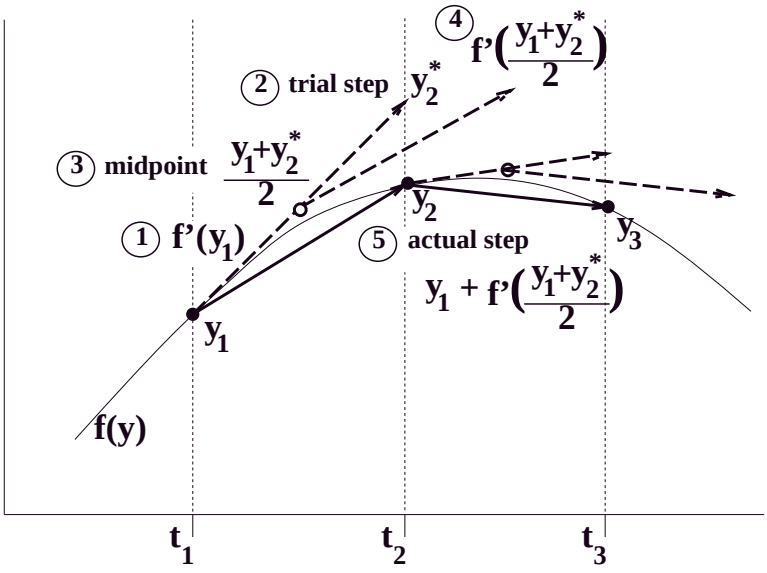

Figure 2.2: The midpoint method. At each point, a trial step is taken along the derivative at that point, and the derivative re-computed at the midpoint between the point and the trial step. This derivative is then used to take the actual step to the next point, and so on.

Further, the consequences of the relationship between CHL as derived from GeneRec and standard error backpropagation (i.e., that CHL uses the faster midpoint integration method and imposes a symmetry-preservation constraint) should be apparent in the relative learning properties of these algorithms. Thus, this derivation might explain why CHL networks tend to learn faster than equivalent backprop networks. Finally, another advantage of a derivation based on the BP framework is that it is sufficiently general as to allow CHL-like learning rules to be derived for a variety of different activation functions or other network properties.

The Midpoint Method and the GeneRec Approximation to it

As was mentioned above, the use of the average of both the minus and plus phase activations of the sending unit in the GeneRec learning rule corresponds to an approximation of a simple numerical integration technique for differential equations known as the midpoint or second-order accurate Runge-Kutta method (Press, Flannery, Teukolsky, & Vetterling, 1988).

The midpoint method attains second-order accuracy without the explicit computation of second derivatives by evaluating the first derivatives twice and combining the results so as to minimize the integration error. It can be illustrated with the following simple differential equation:

$$\frac{dy(t)}{dt} = f'(y(t)) \tag{2.22}$$

The simplest way in which the value of the variable y can be integrated is by using a difference

equation approximation to the continuous differential equation:

$$y_{t+1} = y_t + \epsilon f'(y_t) + O(\epsilon^2) \tag{2.23}$$

with a step size of ϵ, and an accuracy to first order in the Taylor series expansion of $f(y_t)$ (and thus an error term of order ϵ^2). This integration technique is known as the *forward Euler* method, and is commonly used in neural network gradient descent algorithms such as BP. By comparison with (2.23), the midpoint method takes a "trial" step using the forward Euler method, resulting in an estimate of the next function value (denoted by the *):

$$y^*_{t+1} = y_t + \epsilon f'(y_t) \tag{2.24}$$

This estimate is then used to compute the actual step, which is the derivative computed at a point halfway between the current y_t and estimated y^*_{t+1} values (see Figure 2.2):

$$y_{t+1} = y_t + \epsilon f'\left(\frac{y_t + y^*_{t+1}}{2}\right) + O(\epsilon^3) \tag{2.25}$$

In terms of a Taylor series expansion of the function $f(y_t)$ at the point y_t, evaluating the derivative at the midpoint as in (2.25) cancels out the first-order error term ($O(\epsilon^2)$), resulting in a method with second-order accuracy (Press et al., 1988). Intuitively, the midpoint method is able to "anticipate" the curvature of the gradient, and avoid going off too far in the wrong direction.

There are a number of ways the midpoint method could be applied to error backpropagation. Perhaps the most "correct" way of doing it would be to run an entire batch of training patterns to compute the trial step weight derivative, and then run another batch of patterns with the weights half way between their current and the trial step values to get the actual weight changes to be made. However, this would require roughly twice the number of computations per weight update as standard batch-mode backpropagation, and the two passes of batch-mode learning is not particularly biologically plausible.

The GeneRec version of the midpoint method as given by (2.18) is an approximation to the "correct" version in two respects:

1. The plus-phase activation value is used as an on-line estimate of the activations that would result from a forward Euler step over the weights. This estimate has the advantages of being available without any additional computation, and it can be used with on-line weight updating, solving both of the major problems with the "correct" version. The relationship between the plus-phase activation and a forward Euler step along the error gradient makes sense given that the plus-phase activation is the "target" state, which is therefore in the direction of reducing the error. Appendix A gives a more formal analysis of this relationship. This analysis shows that the plus phase activation, which does not depend on the learning rate parameter, typically over-

estimates the size of a trial Euler step. Thus, the use of the plus-phase activation means that the precise midpoint is not actually being computed. Nevertheless, the anticipatory function of this method is still served when the trial step is exaggerated. Indeed, the simulation results described below indicate that it can actually be advantageous in certain tasks to take a larger trial step.

2. The midpoint method is applied only to the portion of the derivative that distributes the unit's error term amongst its incoming weights, and not to the computation of the error term itself. Thus, the error term $(a_j^+ - a_j^-)$ from (2.18) is the same as in the standard forward Euler integration method, and only the sending activations are evaluated at the midpoint between the current and the trial step: $\frac{1}{2}(a_i^- + a_i^+)$. This selective application of the midpoint method is particularly efficient for the case of on-line backpropagation because a midpoint value of the unit error term, especially on a single-pattern basis, will typically be smaller than the original error value, since the trial step is in the direction of reducing error. Thus, using the midpoint error value would actually slow learning by reducing the effective learning rate.

To summarize, the advantage of using the approximate midpoint method represented by (2.18) is that it is so simple to compute, and it appears to reliably speed up learning while still using on-line learning. While other more sophisticated integration techniques have been developed for batch-mode BP (see Battiti, 1992 for a review), they typically require considerable additional computation per step, and are not very biologically plausible.

The GeneRec Approximate Midpoint Method in Backpropagation

In order to validate the idea that CHL is equivalent to GeneRec using the approximation to the midpoint method described above, this approximation can be implemented in a standard backpropagation network and the relative learning speed advantage of this method compared for the two different algorithms. If similar kinds of speedups are found in both GeneRec and backpropagation, this would support the derivation of CHL as given in this paper. Such comparisons are described in the following simulation section.

There are two versions of the GeneRec approximate midpoint method that are relevant to consider. One is a weight-based method which computes the sending unit's trial step activation (h_j^*) based on the trial step weights, and the other is a simpler approximation which uses the unit's error derivative to estimate the trial step activation. In both cases, the resulting trial step activation state h_j^* is averaged with the current activation value h_j to obtain a midpoint activation value which is used as the sending activation state for backpropagation weight updating:

$$\frac{1}{\epsilon}\Delta w_{jk} = \frac{1}{2}(h_j^* + h_j)(t_k - o_k) \tag{2.26}$$

This corresponds to the GeneRec version of the midpoint method as given in (2.18). Note that in a

three-layer network, only the hidden-to-output weights are affected, since there is no trial step activation value for the input units.

In the BP weight-based midpoint approximation, the trial step activation is computed as if the weights had been updated by the trial step weight error derivatives as follows:

$$\begin{aligned} \eta_j^* &= \sum_i s_i \left(w_{ij} + \epsilon_{ts} \frac{-\partial E}{\partial w_{ij}} \right) \\ h_j^* &= \sigma(\eta_j^*) \end{aligned} \tag{2.27}$$

where ϵ_{ts} is a learning-rate-like constant that determines the size of the trial step taken. Note that, in keeping with the GeneRec approximation, the actual learning rate ϵ is not included in this equation. Thus, depending on the relative sizes of ϵ_{ts} and ϵ, the estimated trial step activation given by (2.27) can over-estimate the size of an actual trial step activation. In order to evaluate the effects of this over-estimation, a range of ϵ_{ts} values are explored.

The BP unit-error-based method uses the fact that each weight will be changed in proportion to the derivative of the error with respect to the unit's net input, $\frac{\partial E}{\partial \eta_j}$, to avoid the additional traversal of the weights:

$$\begin{aligned} \eta_j^* &= \eta_j + \epsilon_{ts} F \frac{-\partial E}{\partial \eta_j} \\ h_j^* &= \sigma(\eta_j^*) \end{aligned} \tag{2.28}$$

where F is the number of receiving weights (fan-in). In this case, the trial step size parameter ϵ_{ts} also reflects the average activity level over the input layer, since each input weight would actually be changed by an amount proportional to the activity of the sending unit. The comments regarding ϵ_{ts} above also apply to this case.

Note that it is the unit-error-based version of the midpoint method that most closely corresponds to the version used in CHL, since both are based on the error derivative with respect to the unit, not the weights into the unit. As the simulations reported below indicate, the midpoint method can speed up on-line BP learning by nearly a factor of two. Further, the unit-error-based version is quite simple and requires little extra computation to implement. Finally, while the unit-error-based version could be applied directly to Almeida-Pineda backpropagation, the same is not true for the weight-based version, which would require an additional activation settling based on the trial step weights. Thus, in order to compare these two ways of implementing the approximate midpoint method, the results presented below are for feedforward backprop networks.

Simulation Experiments

The first set of simulations reported in this section is a comparison of the learning speed between several varieties of GeneRec (including symmetric, midpoint, and CHL) and BP with and without the midpoint integration method. This gives a general sense of the comparative learning properties of the different algorithms, and provides empirical evidence in support of the predicted relationships amongst the algorithms investigated. In the second set of simulations, a detailed comparison of the weight derivatives computed by the Almeida-Pineda version of backpropagation and GeneRec is performed, showing that they both compute the same error derivatives under certain conditions.

Learning Speed Comparisons

While it is notoriously difficult to perform useful comparisons between different learning algorithms, such comparisons could provide some empirical evidence necessary for evaluating the theoretical claims made above in the derivation of the GeneRec algorithm and its relationship to CHL. Note that the intent of this comparison is not to promote the use of one algorithm over another, which would require a much broader sample of commonly-used speedup techniques for backpropagation. The derivation of GeneRec based on AP backpropagation and its relationship with CHL via the approximate midpoint method makes specific predictions about which algorithms will learn faster and more reliably than others, and, to the extent that the following empirical results are consistent with these predictions, this provides support for the above analysis. In particular, it is predicted that GeneRec will be able to solve difficult problems in roughly the same order of epochs as the AP algorithm, and that weight symmetry will play an important role in the ability of GeneRec to solve problems. Further, it is predicted that the midpoint versions of both GeneRec and backprop will learn faster than the standard versions.

Overall, the results are consistent with these predictions. It is apparent that GeneRec networks can learn difficult tasks, and further that the midpoint integration method appears to speed up learning in both GeneRec and backpropagation networks. This is consistent with the idea that CHL is equivalent to GeneRec using this midpoint method. Finally, adding the symmetry preservation constraint to GeneRec generally increases the number of networks that solve the task, except in the case of the 4-2-4 encoder for reasons that are explained below. This is consistent with idea that symmetry is important for computing the correct error derivatives.

Four different simulation tasks were studied: XOR (with 2 hidden units), a 4-2-4 encoder, the "shifter" task (Galland & Hinton, 1991), and the "family trees" task of Hinton (1986) (with 18 units per hidden and encoding layer). All networks used 0-to-1 valued sigmoidal units. The backpropagation networks used the cross-entropy error function, with an "error tolerance" of .05, so that if the output activation was within .05 of the target, the unit had no error. In the GeneRec networks, activation values were bounded between .05 and .95. In both the GeneRec and AP backpropagation networks, initial activation values were set to 0, and a step size (dt) of .2 was used to update the ac-

		Euler		Midpoint	
Err Method	FF vs. Rec	NonSym	Sym	NonSym	Sym
BP	FF	BP	—	BP Mid	—
	Rec	AP	—	—	—
Act Diff	FF	—	—	—	—
	Rec	GR	GR Sym	GR Mid	CHL

Table 2.3: Relationship of the algorithms tested with respect to the use of local activations vs. explicit backpropagation (BP, Act Diff) to compute error derivatives, feedforward vs. recurrent (FF, Rec), forward Euler vs. the midpoint method (Euler, Midpoint), and weight symmetrization (NonSym, Sym). GR is GeneRec.

tivations. Settling was stopped when the maximum change in activation (before multiplying by dt) was less than .01. 50 networks with random initial weights (symmetric for the GeneRec networks) were run for XOR and the 4-2-4 encoder, and 10 for the shifter and family trees problems. The training criterion for XOR and the 4-2-4 encoder was .1 total-sum-of-squares error, and the criterion for the shifter and family trees problems was that all units had to be on the right side of .5 for all patterns. Networks were stopped after 5,000 epochs if they had not yet solved the XOR, 4-2-4 encoder and shifter problems, and 1,000 epochs for family trees.

A simple one-dimensional grid search was performed over the learning rate parameter in order to determine the fastest average learning speed for a given algorithm on a given problem. For XOR, the 4-2-4 encoder, and the shifter tasks, the grid was at no less than .05 increments, while a grid of .01 was used for the family trees problem. No momentum or any other modifications to generic backpropagation were used, and weights were updated after every pattern, with patterns presented in a randomly permuted order every epoch. The results presented below are from the fastest networks for which 50% or more did not get stuck in a local minimum. This criterion is really only important for the XOR problem, since the algorithms did not typically get stuck on the other problems.

The algorithms compared were as follows (see table 2.3):

BP Standard feedforward error backpropagation using the cross-entropy error function.

AP Almeida-Pineda backpropagation in a recurrent network using the cross-entropy error function.

BP Mid Wt Feedforward error backpropagation with the weight-based version of the approximate midpoint (2.27). Several different values of the trial step size parameter ϵ_{ts} were used in order to determine the effects of overestimating the trial step as is the case with GeneRec. The values were: 1, 5, 10, and 25 for XOR and the 4-2-4 encoder, .5, 1, and 2 for the shifter problem, and .05, .1, .2, and .5 for the family trees problem. The large trial step sizes resulted in faster learning in small networks, but progressively smaller step sizes were necessary for the larger problems.

BP Mid Un Feedforward error backpropagation with the unit-error based version of the midpoint integration method (2.28). The same trial step size parameters as in BP Mid Wt were used.

Algorithm	ϵ	N	Epcs	SEM
BP	1.95	37	305	58
AP	1.40	35	164	23
BP Mid Wt 1	1.85	39	268	59
BP Mid Wt 5	0.25	25	326	79
BP Mid Wt 10	0.25	34	218	25
BP Mid Wt 25	0.35	27	215	40
BP Mid Un 1	1.40	40	222	28
BP Mid Un 5	1.05	34	138	38
BP Mid Un 10	0.40	26	222	10
BP Mid Un 25	0.30	31	178	37
GR	0.20	9†	3795	267
GR Sym	0.60	31	334	7.1
GR Mid	1.75	33	97	4.6
CHL	1.80	28	59	1.8

Table 2.4: Results for the XOR problem. ϵ is the optimal learning rate, *N* is the number of networks that successfully solved the problem (out of 50, minimum of 25), *Epcs* is the mean number of epochs required to reach criterion, and *SEM* is the standard error of this mean. Algorithms are as described in the text. † Note that this was the best performance for the GR networks.

GR The basic GeneRec algorithm (2.17).

GR Sym GeneRec with the symmetry preservation constraint (2.20).

GR Mid GeneRec with the approximate midpoint method (2.18).

CHL GeneRec with both symmetry and approximate midpoint method, which is equivalent to CHL (2.7).

XOR and the 4-2-4 Encoder

The results for the XOR problem are shown in table 2.4, and those for the 4-2-4 encoder are shown in table 2.5. These results are largely consistent with the predictions made above, with the exception of an apparent interaction between the 4-2-4 encoder problem and the use of weight symmetrization in GeneRec. Thus, it is apparent that the plain GeneRec algorithm is not very successful or fast, and that weight symmetrization is necessary to improve the success rate (in the XOR task) and the learning speed (in the 4-2-4 encoder). As will be shown in more detail below, the symmetrization constraint is essential for computing the correct error derivatives in GeneRec.

However, the symmetry constraint also effectively limits the range of weight space that can be searched by the learning algorithm (only symmetric weight configurations can be learned), which might affect its ability to get out of bad initial weight configurations. This effect may be compounded in an encoder problem, where the input-to-hidden weights also have a tendency to become symmetric with the hidden-to-output weights. Thus, while the symmetry constraint is important for being able to compute the correct error derivatives, it also introduces an additional constraint which can impair

Algorithm	ϵ	N	Epcs	SEM
BP	2.40	50	60	5.1
AP	2.80	50	54	3.6
BP Mid Wt 1	1.70	50	60	4.3
BP Mid Wt 5	1.65	50	48	2.8
BP Mid Wt 10	2.35	50	45	3.6
BP Mid Wt 25	2.25	50	37	3.0
BP Mid Un 1	2.20	50	54	4.2
BP Mid Un 5	2.10	50	42	2.5
BP Mid Un 10	2.10	50	40	2.9
BP Mid Un 25	1.95	50	34	1.8
GR	0.60	45	418	28
GR Sym	1.40	28	88	2.9
GR Mid	2.40	46	60	3.4
CHL	1.20	28	77	1.8

Table 2.5: Results for the 4-2-4 encoder problem. ϵ is the optimal learning rate, *N* is the number of networks that successfully solved the problem (out of 50), minimum of 25, *Epcs* is the mean number of epochs required to reach criterion, and *SEM* is the standard error of this mean. Algorithms are as described in the text.

learning, sometimes dramatically (as in the case of the 4-2-4 encoder). Note that on larger and more complicated tasks like the shifter and family trees described below, the advantages of computing the correct derivatives begin to outweigh the disadvantages of the additional symmetry constraint.

The other main prediction from the analysis is that the approximate midpoint method will result in faster learning, both in BP and GeneRec. This appears to be the case, where the speedup relative to regular backprop was nearly two-fold for the unit-error based version with a trial step size of 25. The general advantage of the unit-error over the weight based midpoint method in BP is interesting considering that this corresponds to the GeneRec version of the midpoint method. The speedup in GeneRec for both the CHL *vs* GR Sym and GR Mid *vs* GR comparisons was substantial in general. Further, it is interesting that the approximate midpoint method alone (without the symmetrization constraint) can enable the GeneRec algorithm to successfully solve problems. Indeed, on both of these tasks, GR Mid performed better than GR Sym. This might be attributable to the ability of the midpoint method to compute better weight derivatives which are less affected by the inaccuracies introduced by the lack of weight symmetry. However, note that while this seems to hold for all of the three-layer networks studied, it breaks down in the family trees task which requires error derivatives to be passed back through multiple hidden layers. Also, only on the 4-2-4 encoder did GR Mid perform better than CHL, indicating that there is generally an advantage to having the correct error derivatives via weight symmetrization in addition to using the midpoint method.

An additional finding is that there appears to be an advantage for the use of a recurrent network over a feedforward one, based on a comparison of AP *vs* BP results. This can be explained by the fact that small weight changes in a recurrent network can lead to more dramatic activation state differences than in a feedforward network. In effect, the recurrent network has to do less work to achieve a

Algorithm	ϵ	N	Epcs	SEM
BP	1.25	10	76.2	6.4
AP	1.35	10	56.8	4.2
BP Mid Wt .5	0.40	10	63.6	4.8
BP Mid Wt 1	0.45	10	42.5	2.9
BP Mid Wt 2	0.35	10	47.0	3.5
BP Mid Un .5	0.30	10	48.0	1.7
BP Mid Un 1	0.35	10	41.2	3.3
BP Mid Un 2	0.15	10	51.8	3.8
GR	0.10	1	1650	—
GR Sym	0.90	10	105	5.0
GR Mid	0.65	10	84.2	13.4
CHL	0.70	10	42.7	2.2

Table 2.6: Results for the shifter task. ϵ is the optimal learning rate, *N* is the number of networks that successfully solved the problem (out of 10), *Epcs* is the mean number of epochs required to reach criterion, and *SEM* is the standard error of this mean. Algorithms are as described in the text.

given set of activation states than does a feedforward network. This advantage for recurrency, which should be present in GeneRec, is probably partially offset by the additional weight symmetry constraint. Further, recurrency appears to become a liability in networks with multiple hidden layers, based on the family trees results presented below.

The Shifter Task

The shifter problem is a larger task than XOR and the 4-2-4 encoder, and thus might provide a more realistic barometer of performance on typical tasks.[5] The version of the shifter problem used here had two 4 bit input patterns, one of which was a shifted version of the other. There were three values of shift, -1, 0, and 1, corresponding to one bit to the left, the same, and one bit to the right (with wrap-around). Of the 16 possible binary patterns on 4 bits, 4 were unsuitable because they result in the same pattern when shifted right or left (1111, 1010, 0101, and 0000). Thus, there were 36 training patterns (the 12 bit patterns shifted in each of 3 directions). The task was to classify the shift direction by activating one of 3 output units. While larger versions of this task (more levels of shift, more bits in the input) were explored, this configuration proved the most difficult (in terms of epochs) for a standard BP network to solve.

The results, shown in table 2.6, provide clearer support for the predicted relationships than the two previous tasks. In particular, the midpoint-based speedup is comparable between the BP and GeneRec cases, and the role of symmetry in GeneRec is unambiguously important for solving the task, as is evident from the almost complete failure of the non-symmetric version to learn the problem. However, it is interesting that even in this more complicated problem the use of the approximate midpoint method without the additional symmetrizing constraint enables the GeneRec networks to

[5] Note that other common tasks like digit recognition or other classification tasks were found to be so easily solved by a standard BP network (under 10 epochs), that they did not provide a useful dynamic range to make the desired comparisons.

Algorithm	ϵ	N	Epcs	SEM
BP	0.39	10	129	3.0
AP	0.30	10	181	11
BP Mid Wt .05	0.37	10	131	6.4
BP Mid Wt .1	0.38	10	130	5.1
BP Mid Wt .2	0.21	10	136	6.0
BP Mid Un .05	0.24	10	127	6.4
BP Mid Un .1	0.23	10	114	6.7
BP Mid Un .2	0.19	10	123	8.7
GR	—	0	—	—
GR Sym	0.20	10	409	14
GR Mid	—	0	—	—
CHL	0.10	10	328	23

Table 2.7: Results for the family trees problem. ϵ is the optimal learning rate, *N* is the number of networks that successfully solved the problem (out of 10, minimum of 5), *Epcs* is the mean number of epochs required to reach criterion, and *SEM* is the standard error of this mean. Algorithms are as described in the text.

learn the problem. Nevertheless, the combination of the approximate midpoint method and the symmetrizing constraint (i.e., the CHL algorithm) performs better than either alone.

As in the previous tasks, there appears to be an advantage for the use of a recurrent network over a non-recurrent one, as evidenced by the faster learning of AP compared to BP.

The Family Trees Task

As was mentioned in the introduction, the family trees problem Hinton (1986) is of particular interest because Galland (1993) reported that he was unable to train CHL to solve this problem. While I was unable to get a CHL network to learn the problem with the same number of hidden units as was used in the original backprop version of this task (6 "encoding" units per input/output layer, and 12 central hidden units), simply increasing the number of encoding units to 12 was enough to allow CHL to learn the task, although not with 100% reliability. Thus, the learning rate search was performed on networks with 18 encoding and 18 hidden units to ensure that networks were capable of learning.

As can be seen from the results shown in table 2.7, the CHL networks were able to reliably solve this task within a roughly comparable number of epochs as the AP networks. Note that the recurrent networks (GeneRec and AP) appear to be at a disadvantage relative to feedforward BP[6] on this task, probably due to the difficulty of shaping the appropriate attractors over multiple hidden layers. Also, symmetry preservation appears to be critical for GeneRec learning in deep networks, since GeneRec networks without this were unable to solve this task (even with the midpoint method).

[6] It should be noted that the performance of feedforward BP on this task is much faster than previously reported results. This is most likely due to the use of on-line learning and not using momentum, which enables the network to take advantage of the noise due to the random order of the training patterns to break the symmetry of the error signals generated in the problem and distinguish amongst the different training patterns.

The comparable performance of AP and CHL supports the derivation of CHL via the GeneRec algorithm as essentially a form of backpropagation, and calls into question the analyses of Galland (1993) regarding the limitations of CHL as a deterministic approximation to a Boltzmann machine. It is difficult to determine what is responsible for the failure to learn the family trees problem reported in Galland (1993), since there are several differences in the way those networks were run compared to the ones described above, including the use of an annealing schedule, not using the .05, .95 activation cutoff, using activations with -1 to +1 range, using batch mode instead of on-line weight updating, and using activation-based as opposed to net-input-based settling.

Finally, only the unit-error based midpoint method in backpropagation showed a learning speed advantage in this task. This is consistent with the trend of the previous results. The advantage of the unit-error based midpoint method might be due to the reliance on the derivative of the error with respect to the hidden unit itself, which could be a more reliable indication of the curvature of the derivative than the weight derivatives used in the other method.

The GeneRec Approximation to AP BP

The analysis presented earlier in the paper shows that GeneRec should compute the same error derivatives as the Almeida-Pineda version of error backpropagation in a recurrent network if the following conditions hold:

- The difference of the plus and minus phase activation terms in GeneRec, which are updated in separate iterative activation settling phases, can be used to compute a unit's error term instead of the iterative update of the difference itself, which is what Almeida-Pineda uses.
- The reciprocal weights are symmetric. This enables the activation signals from the output to the hidden units (via the recurrent weights) to reflect the contribution that the hidden units made to the output error (via the forward-going weights).
- The difference of activations in the plus and minus phases is a reasonable approximation to the difference of net inputs times the derivative of the sigmoidal activation function. Note that this only affects the overall magnitude of the weight derivatives, not their direction.

In order to evaluate the extent to which these conditions are violated and the effect that this has on learning in GeneRec, two identical networks were run side-by-side on the same sequence of training patterns, with one network using AP (with the cross-entropy error function) to compute the error derivatives, and the other using GeneRec. The standard 4-2-4 encoder problem was used. The extent to which GeneRec error derivatives are the same as those computed by AP was measured by the normalized dot product between the weight derivative vectors computed by the two algorithms. This comparison was made for the input-to-hidden weights ($I \Rightarrow H$) since they reflect the error derivatives computed by the hidden units. Since the hidden-to-output weights are driven by the error signal

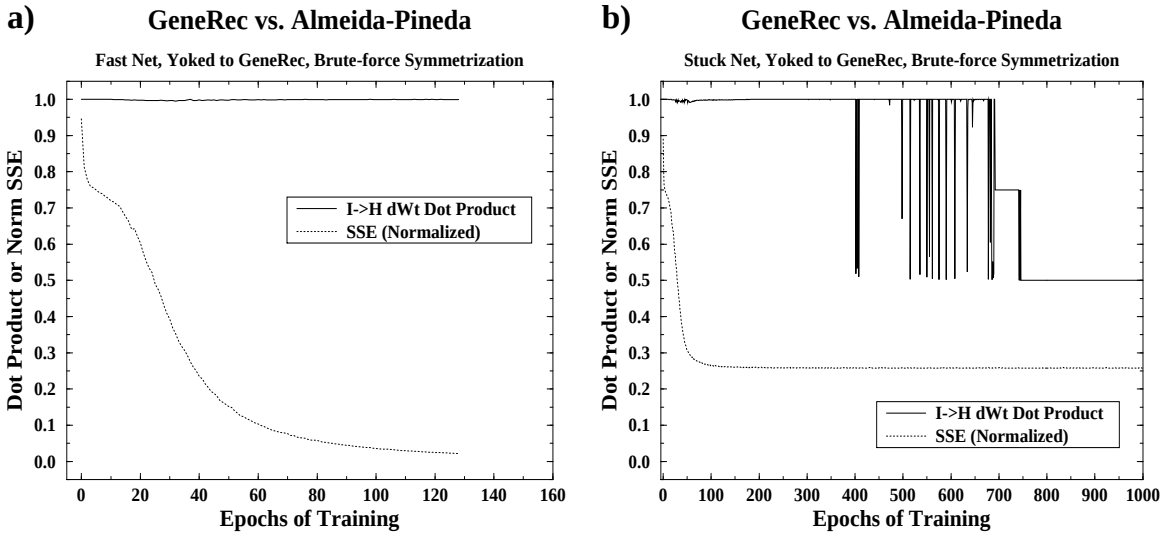

Figure 2.3: Correspondence (average normalized dot product) of weight derivatives computed by GeneRec and Almeida-Pineda algorithms for two random initial weight configurations. The weights were yoked to those computed by GeneRec, and symmetry was preserved by the brute force method. The correspondence is nearly perfect until late in the training of the stuck network, at which point the network has developed very large weights, which appear to affect the accuracy of the computed weight derivatives.

on the output units, which is given by the environment, these derivatives were always identical between the two networks. In order to control for weight differences that might accumulate over time in the two networks, the weights were copied from the GeneRec network to the AP network after each weight update. Networks were also run without this "yoking" of the weights in order to determine how different the overall learning trajectory was between the two algorithms given the same initial weight values. The weights were always initialized to be symmetric.

As was noted above, the basic GeneRec algorithm does not preserve the symmetry of the weights, which will undoubtedly affect the computation of error gradients. The extent of symmetry was measured by the normalized dot product between the reciprocal hidden and output weights. It is predicted that this symmetry measure will determine in large part the extent to which GeneRec computes the same error derivatives as AP. In order to test this hypothesis, two methods for preserving the symmetry of the weights during learning were also used. One method was to use the symmetry-preserving learning rule shown in (2.20), and the other was a "brute-force" method where reciprocal weights were set to the average of the two values after they were updated. The advantage of this later method is that, unlike the (2.20) rule, it does not change the computed weight changes.

The parameters used in the networks were: activation step size (dt) of .2, initial activations set to 0, settling cutoff at .01 maximum change in activation, learning rate of .6, and initial weights uniformly random between $\pm .5$.

The main result of this analysis is that the GeneRec algorithm typically computes essentially the same error derivatives as AP *except when the weights are not symmetric*. This can be seen in Figure 2.3, which shows two different networks running with weights yoked and using the brute-force

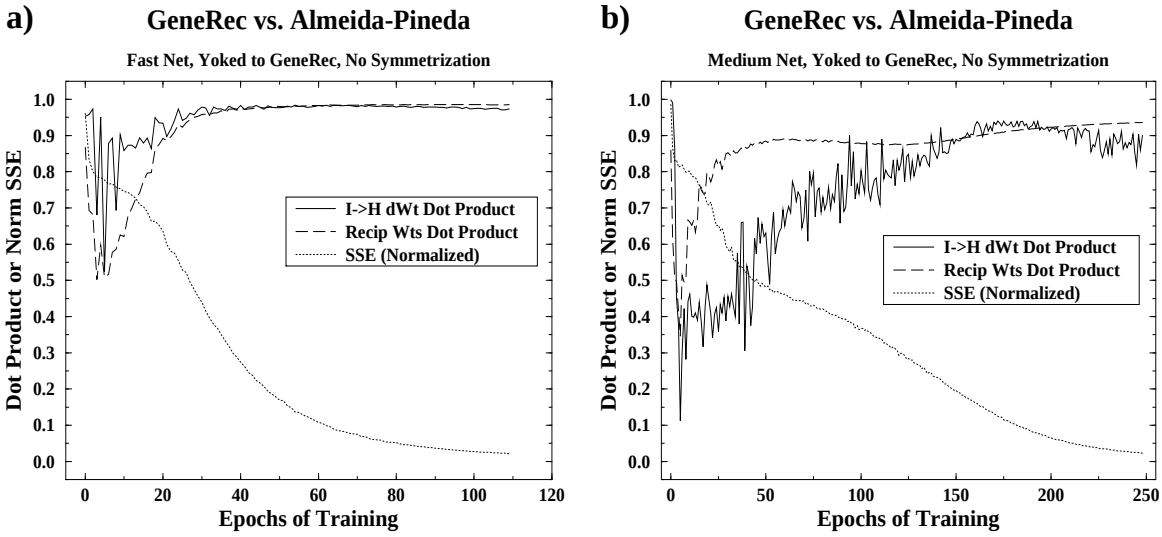

Figure 2.4: Correspondence (average normalized dot product) of weight derivatives computed by GeneRec and Almeida-Pineda algorithms for two random initial weight configurations. The weights were yoked to those computed by GeneRec. No symmetry was imposed on the weights. The correspondence appears to be roughly correlated with the extent to which the weights are symmetric.

symmetrizing method. The weight derivatives computed by GeneRec have a normalized dot product with those computed by AP that is nearly always 1, except when the weights got very large in the network that was stuck in a local minimum. This result shows that GeneRec usually computes the appropriate backpropagation error gradient based on the difference of equilibrium activation states in the plus and minus phases, supporting the approximation given in (2.16).

In contrast, when no weight symmetrizing is being enforced, the correspondence between the GeneRec and AP weight derivatives appears to be correlated with the extent to which the weights are symmetric, as can be seen in Figure 2.4. Indeed, based on results of many runs (not shown), the ability of the GeneRec network to solve the task appeared to be correlated with the extent to which the weights remain symmetric. Note that even without explicit weight symmetrization or a symmetry preserving learning rule, the weights can become symmetric due to a fortuitous correspondence between weight changes on the reciprocal sets of weights.

Using the symmetry preserving rule (2.20) resulted in weight changes that were typically different from those computed by AP, even though the above results show that the error derivatives at the hidden unit were correct. This is simply due to the fact that symmetric GeneRec has an additional symmetry preserving term which is not present in AP. Nevertheless, the symmetric GeneRec algorithm resulted in non-yoked learning trajectories which mirrored those of the AP algorithm remarkably closely. A representative example is shown in Figure 2.5. It is difficult to be certain about the source of this correspondence, which did not occur in non-yoked networks using the brute-force symmetry preservation method.

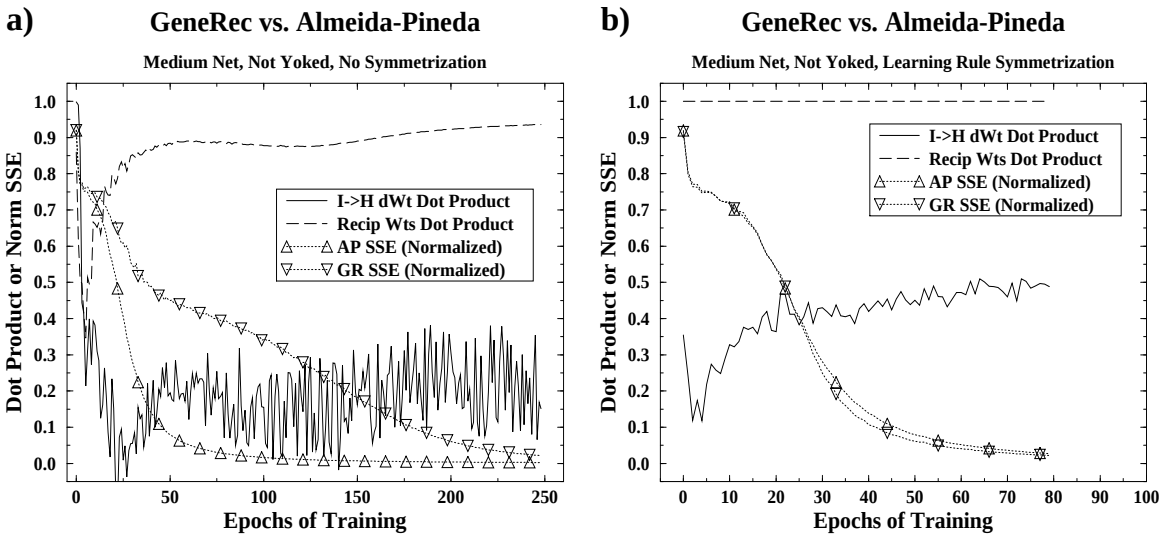

Figure 2.5: Learning trajectories and error derivative correspondence for non-yoked AP and GeneRec networks with the same initial weights. **a)** Shows standard GeneRec without any weight symmetrization. **b)** shows GeneRec with the symmetry preserving learning rule. Even though this rule does not result in the networks computing the same weight updates, they follow a remarkably similar learning trajectory. This is not the case for regular GeneRec.

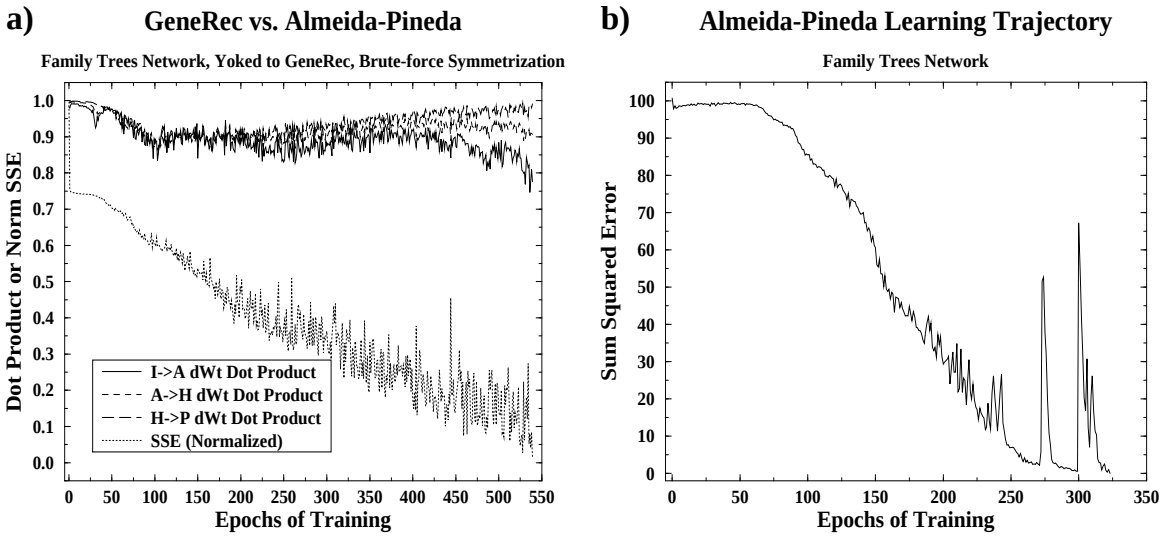

Figure 2.6: **a)** Correspondence (average normalized dot product) of weight derivatives computed by GeneRec and Almeida-Pineda algorithms for a family trees network. The weights were yoked to those computed by GeneRec, and symmetry was preserved by the brute force method. $I \Rightarrow A$ is the "agent input" to the "agent encoding" hidden layer weights, $A \Rightarrow H$ is the "agent encoding" to the central hidden layer weights, and $H \Rightarrow P$ is the hidden layer to the "patient encoding" layer weights. The correspondence is not as good as a that for the three layer network, but remains largely above .9. **b)** Shows the learning trajectory for just the AP algorithm, which is not smooth like feedforward BP.

Finally, there is some question as to whether GeneRec will compute the correct error derivatives in a network with multiple hidden layers, where the differences in the way GeneRec and AP compute the error terms might become more apparent due to the greater influence of recurrent setting in both the minus and plus phases. Also, based on the kinds of approximations made in deriving CHL as a deterministic Boltzmann machine, and the results of simulations, Galland (1993) concluded that the limitations of CHL become more apparent as the number of hidden layers are increased (i.e., in "deep" networks).

To address the performance of GeneRec in a deep network, the same analysis as described above was performed on the family trees network (Hinton, 1986) with the brute-force symmetrization and weight yoking. This network has three layers of hidden units. Normalized dot-product measurements of the error derivatives computed on the weights from the "agent input" to the "agent encoding" hidden layer $I \Rightarrow A$, "agent encoding" to the central hidden layer $A \Rightarrow H$, and the hidden layer to the "patient encoding" layer $H \Rightarrow P$. These weights are 3, 2, and 1 hidden layers (respectively) removed from the output layer. Figure 2.6a shows that GeneRec still computes largely the same error derivatives as AP backpropagation even in this case. The normalized dot product measures were usually greater than .9, and never went below .7. The discrepancy between GeneRec and AP tended to increase as training proceeded for the deeper weights ($I \Rightarrow A$). This shows that as the weights got larger, the differences between GeneRec and AP due to the way that the error is computed over recurrent settling became magnified.

One of the primary problems with CHL that was emphasized by Galland (1993) is the jumpy character of the error function over learning, which was argued to not provide much useful guidance in learning. However, Figure 2.6b shows that the AP algorithm also suffers from a bumpy error surface. The frequency of the AP bumps seems to be a bit lower, but the amplitude can be higher. This indicates that the bumpiness is due to the recurrent nature of the network, where small weight changes can lead to very different activation states, and not to a deficiency in the learning algorithm *per se*.

Possible Biological Implementation of GeneRec Learning

The preceding analysis and simulations show that the GeneRec family of phase-based, error-driven learning rules can approximate error backpropagation using locally available activation variables. The fact that these variables are available locally makes it more plausible that such a learning rule could be employed by real neurons. Also, the use of activation-based signals (as opposed to error or other variables) increases plausibility because it is relatively straightforward to map unit activation onto neural variables such as time-averaged membrane potential or spiking rate. However, there are three main features of the GeneRec algorithm that could potentially be problematic from a biological perspective: 1) weight symmetry; 2) the origin of plus and minus phase activation states; 3) the ability of these activation states to influence synaptic modification according to the learning

rule. These issues are addressed below in the context of the CHL version of GeneRec (2.7), since it is the overall best performer, and has a simpler form than the other GeneRec versions. Since the neocortex is the single most important brain area for the majority of cognitive phenomena, it is the focus of this discussion.

Weight Symmetry in the Cortex

There are two ways in which the biological plausibility of the weight symmetry requirement in GeneRec (which was shown above to be important for computing the correct error gradient) can be addressed. One is to show that exact symmetry is not critical to the proper functioning of the algorithm, so that only a rough form of symmetry would be required of the biology. The other is to show that at least this rough form of symmetry is actually present in the cortex. Data consistent with these arguments are summarized briefly here.

As a first-order point, Hinton (1989b) noted that a symmetry preserving learning algorithm like CHL, when combined with weight decay, will automatically lead to symmetric weights even if they did not start out that way. However, this assumes that all of the units are connected to each other in the first place. This more difficult case of connection asymmetry was investigated in Galland and Hinton (1991) for the CHL learning algorithm. It was found that the algorithm was still effective even when all of the connectivity was asymmetric (i.e., for each pair of non-input units, only one of the two possible connections between them existed). This robustness can be attributed to a redundancy in the ways in which the error signal information can be obtained (i.e., a given hidden unit could obtain the error signal directly from the output units, or indirectly through connections to other hidden units). Also, note that the absence of any connection at all is very different from the presence of connection with a non-symmetric weight value, which is the form of asymmetry that was found to be problematic in the above analysis. In the former case, only a subset of the error gradient information is available, while the latter case can result in specifically *wrong* gradient information due to the influence of the non-symmetric weight. Due to the automatic symmetrization property of CHL, the latter case is unlikely to be a problem.

In terms of biological evidence for symmetric connectivity, there is some indication that the cortex is at least roughly symmetrically connected. At the level of identifiable anatomical subregions of cortex, the vast majority of areas (at least within the visual cortex) are symmetrically connected to each other. That is, if area A projects to area B, area A also receives a projection from area B (Felleman & Van Essen, 1991). At the level of cortical columns or "stripes" within the prefrontal cortex of the monkey, Levitt, Lewis, Yoshioka, and Lund (1993) showed that connectivity was symmetric between interconnected stripes. Thus, if a neuron received projections from neurons in a given stripe, it also projected to neurons in that stripe. The more detailed level of individual neuron symmetric connectivity is difficult to assess empirically, but there is at least no evidence that it does *not* exist. Further, given that there is evidence for at least rough symmetry, detailed symmetry may not be critical since, as demonstrated by Galland and Hinton (1991), CHL can use asymmetrical connec-

tivity as long as there is some way of obtaining reciprocal information through a subset of symmetric connections or indirectly via other neurons in the same area.

Phase-Based Activations in the Cortex

The origin of the phase-based activations that are central to the GeneRec algorithm touches at the heart of perhaps the most controversial aspect of error-driven learning in the cortex: "where does the teaching signal come from?" In GeneRec, the teaching signal is just the plus-phase activation state. Thus, unlike standard backpropagation, GeneRec suggests that the teaching signal is just another state of "experience" in the network. One can interpret this state as that of experiencing the actual *outcome* of some previous conditions. Thus, the minus phase can be thought of as the *expectation* of the outcome given these conditions. For example, after hearing the first three words of a sentence, an expectation will develop of which word is likely to come next. The state of the neurons upon generating this expectation is the minus phase. The experience of hearing or reading the actual word that comes next establishes a subsequent locomotive[7] state of activation, which serves as the plus phase. This idea that the brain is constantly generating expectations about subsequent events, and that the discrepancies between these expectations and subsequent outcomes can be used for error-driven learning, has been suggested by McClelland (1994) as a psychological interpretation of the backpropagation learning procedure. It is particularly attractive for the GeneRec version of backpropagation, which uses only activation states, because it requires no additional mechanisms for providing specific teaching signals other than the effects of experience on neural activation states in a manner that is widely believed to be taking place in the cortex anyway.

Further, there is evidence from ERP recordings of electrical activity over the scalp during behavioral tasks that cortical activation states reflect expectations and are sensitive to differential outcomes. For example, the widely-studied *P300* wave, which is a positive-going wave that occurs around 300 msec after stimulus onset, is considered to measure a violation of subjective expectancy which is determined by preceding experience over both the short and long term (Hillyard & Picton, 1987). In more formal terms, Sutton, Braren, Zubin, and John (1965) showed that the P300 amplitude is determined by the amount of prior uncertainty that is resolved by the processing of a given event. Thus, the nature of the P300 is consistent with the idea that it represents a plus phase wave of activation following in a relatively short time-frame the development of minus phase expectations. While the specific properties of the P300 itself might be due to specialized neural mechanisms for monitoring discrepancies between expectations and outcomes, its presence suggests the possibility that neurons in the mammalian neocortex experience two states of activation in relatively rapid succession, one corresponding to expectation and the other corresponding to outcome.

Finally, note that for most of the GeneRec variants, it seems that the neuron needs to have both the plus and minus phase activation signals in reasonably close temporal proximity in order to adjust

[7] This is just to demonstrate that such expectations are being generated and it is salient when they are violated.

Plus Phase Variables	Minus Phase Variables	
	$a_i^- a_j^- \approx 0$ ($[Ca^{2+}]_i$ near 0)	$a_i^- a_j^- \approx 1$ ($[Ca^{2+}]_i$ elevated)
$a_i^+ a_j^+ \approx 0$	$\Delta w_{ij} = 0$	$\Delta w_{ij} = -$ (LTD)
$a_i^+ a_j^+ \approx 1$	$\Delta w_{ij} = +$ (LTP)	$\Delta w_{ij} = 0^*$

Table 2.8: Directions of weight change according to the CHL rule for four qualitative conditions, consisting of the combinations of two qualitative levels of minus and plus phase activation coproduct values. The minus phase activation coproduct is thought to correspond to $[Ca^{2+}]_i$. Increases in synaptic efficacy correspond to long-term potentiation (LTP) and decreases are long-term depression (LTD). * This cell is not consistent with the biological mechanism because both $[Ca^{2+}]_i$ and synaptic activity lead to LTP, not the absence of LTP. See text for a discussion of this point.

its synapses based on both values. This is consistent with the relatively rapid expectation-outcome interpretation given above. However, CHL is a special case, since it is simply the difference between the coproduct of same-phase activations, which could potentially be computed by performing simple Hebbian associative learning for the plus phase at any point, and at any other point, performing anti-Hebbian learning on the minus phase activations (Hinton & Sejnowski, 1986). This leaves open the problems of how the brain would know when to change the sign of the weight change, and how this kind of global switch could be implemented. Also, people are capable of learning things relatively quickly (within seconds or at least minutes), so this phase switching is unlikely to be a function of the difference between REM sleep and waking behavior, as has been suggested for phase-based learning algorithms (Hinton & Sejnowski, 1986; Linsker, 1992; Crick & Mitchison, 1983). While it might be possible to come up with answers to these problems, a temporally local mechanism like that suggested above seems more plausible.

Synaptic Modification Mechanisms

Having suggested that the minus and plus phase activations follow each other in rapid succession, it remains to be shown how these two activation states could influence synaptic modification in a manner largely consistent with the CHL version of GeneRec (2.7). It turns out that the biological mechanism proposed below accounts for only three out of four different qualitative ranges of the sign of the weight change required by CHL (see table 2.8). Specifically, the proposed mechanism predicts weight increase to occur when both the pre and postsynaptic neurons are active in both the plus and minus phases, whereas CHL predicts that the weight change in this condition should be zero. Thus, the proposed mechanism corresponds to a combination of CHL and a Hebbian-style learning rule, the computational implications of which are the subject of the remainder of this thesis, which shows that the combination of error-driven and associative learning can be generally beneficial for solving many different kinds of tasks. However, for the purposes of the present paper, the crucial aspect of the following mechanism is that it provides the error correction term, which occurs when the synaptic coproduct $a_i a_j$ was larger in the minus phase than in the plus phase. This is the defining aspect of the error-driven learning performed by CHL, since the other qualitative ranges of the CHL learning rule are similar to standard Hebbian learning, as is evident from table 2.8.

For GeneRec-style learning to occur at a cellular and synaptic level, the neuron needs to be able to retain some trace of the minus phase activation state through the time when the neuron experiences its plus phase activation state. Reasoning from the ERP data described above, this time period might be around 300 milliseconds or so. A likely candidate for the minus phase trace is intracellular Ca^{2+} ($[Ca^{2+}]_i$), which enters the postsynaptic area via NMDA channels if both pre and postsynaptic neurons are active. To implement a GeneRec-style learning rule, this minus phase $[Ca^{2+}]_i$ trace needs to interact with the subsequent plus phase activity to determine if the synapse is potentiated (LTP) or depressed (LTD). In what follows, the term *synaptic activity* will be used to denote the activation coproduct term $a_i a_j$, which is effectively what determines the amount of $[Ca^{2+}]_i$ that enters through the NMDA channel (Collingridge & Bliss, 1987).

There are two basic categories of mechanism which can provide the crucial error-correcting modulation of the sign of synaptic modification required by CHL. One such mechanism involves an interaction between membrane potential or synaptic activity and $[Ca^{2+}]_i$, while another depends only on the level of $[Ca^{2+}]_i$. Further, there are many ways in which these signals and their timing can affect various second-messenger systems in the cell to provide the necessary modulation. In favor of something like the first mechanism, there is evidence that the mere presence of postsynaptic $[Ca^{2+}]_i$ is insufficient to cause LTP (Kullmann, Perkel, Manabe, & Nicoll, 1992; Bashir, Bortolotto, & Davies, 1993), but it is unclear exactly what additional factor is necessary (Bear & Malenka, 1994). One hypothesis is that LTP depends on the activation of metabotropic glutamate receptors, which are activated by presynaptic activity and can trigger various mechanisms in the postsynaptic synaptic compartment (Bashir et al., 1993). On the other hand, a proposed mechanism that depends only on the level of postsynaptic $[Ca^{2+}]_i$ (Lisman, 1989), has received some recent empirical support (reviewed in Lisman, 1994; Bear & Malenka, 1994). This proposal stipulates that increased but moderate concentrations of postsynaptic $[Ca^{2+}]_i$ lead to LTD, while higher concentrations lead to LTP. Artola and Singer (1993) argue that this mechanism is consistent with the *ABS* learning rule (Hancock, Smith, & Phillips, 1991; Artola, Brocher, & Singer, 1990; Bienenstock, Cooper, & Munro, 1982), which stipulates that there are two thresholds for synaptic modification, Θ^+ and Θ^-. A level of $[Ca^{2+}]_i$ which is higher than the high threshold Θ^+ leads to LTP, while a level which is lower than this high threshold but above the lower Θ^- threshold leads to LTD.

Either of the above mechanisms would be capable of producing the pattern of synaptic modification shown in table 2.8 in the context of a proposed mechanism defined by the following properties:

1. Some minimal level of $[Ca^{2+}]_i$ is necessary for any form of synaptic modification (LTP or LTD).

2. $[Ca^{2+}]_i$ changes relatively slowly, and persists for at least 300+ milliseconds. This allows $[Ca^{2+}]_i$ to represent prior minus phase activity, even if the synapse is not subsequently active in the plus phase.

3. Synaptic modification occurs based on the postsynaptic state after plus phase activity. This can

happen locally if synaptic modification occurs after around 300+ milliseconds since the entry of Ca^{2+} into the postsynaptic area (and the plus phase activity states last for at least this long). Alternatively, there could be a relatively global signal corresponding to the plus phase that triggers synaptic modification (e.g., as provided by dopaminergic or cholinergic modulation triggered by systems sensitive to the experience of outcomes following expectations).

4. If $[Ca^{2+}]_i$ was present initially (in the minus phase) due to synaptic activity, but then the synaptic activity diminished or ceased (in the plus phase), LTD should occur. This would be expected from the mechanisms described above, either because of an explicit interaction between synaptic activity at the time of modification in the plus phase and the trace of $[Ca^{2+}]_i$ from the minus phase, or because the minus phase $[Ca^{2+}]_i$ will have decayed into the LTD range by the time modification occurs in the plus phase.

5. If synaptic activity is taking place in the plus phase state, sufficient $[Ca^{2+}]_i$ is present and LTP occurs. Note that this means that any time the plus-phase activation coproduct ($a_i^+ a_j^+$) is reasonably large, regardless of whether there was any prior minus phase activity, the weights will be increased. This leads to a combined CHL and Hebbian learning rule as discussed above.

There is direct evidence in support of several aspects of the proposed mechanism, some of which was discussed above, and indirect evidence in support of most of the remainder. Since the empirical literature on LTP and LTD is vast, only a brief summary will be given here (see Artola & Singer, 1993; Bear & Malenka, 1994; Linden, 1994; Malenka & Nicoll, 1993, for recent reviews). It should be noted that most of these findings have been described both in the hippocampus and neocortex, and appear to be quite general (Artola & Singer, 1993; Linden, 1994). Also, note that the NMDA receptor itself is not subject to potentiation, so that the current value of the synaptic weight does not have to be included in learning rules, which is in accordance with GeneRec.

With respect to point 1, the importance of $[Ca^{2+}]_i$ for LTP has been known for a while (e.g., Collingridge & Bliss, 1987), and it is now clear that it is critical for LTD as well (Brocher, Artola, & Singer, 1992; Mulkey & Malenka, 1992; Hirsh & Crepel, 1992). In support of point 2, the time course of $[Ca^{2+}]_i$ concentration has been measured in several studies (e.g., Jaffe, Johnston, Lasser-Ross, Lisman, Miyakawa, & Ross, 1992; Perkel, Petrozzino, Nicoll, & Connor, 1993), and it appears to be relatively long-lasting (on the order of 1 or more seconds), though it is not clear that these results reflect what would happen under less invasive conditions.

As for point 3, Malenka, Lancaster, and Zucker (1992) found that a significant time period (up to 1-2 seconds) of enhanced postsynaptic $[Ca^{2+}]_i$ was necessary for LTP induction. Also, typical LTP and LTD induction regimes involve constant stimulation at a given frequency for time periods longer than a second. However, the precise time course of synaptic potentiation needs to be studied in greater detail to evaluate this issue fully. With respect to the existence of a global learning signal, the ERP data described earlier and the role of neuromodulatory systems like dopamine suggest that such

monitoring systems might exist in the brain. For example, Schultz, Apicella, and Ljungberg (1993) describe the important role that dopamine plays in learning and responding to salient environmental stimuli. However, these modulatory effects are probably not of an all-or-nothing nature, and, given that LTP and LTD can be induced by the direct electrical stimulation of individual neurons, it is not likely that learning is completely dependent on a global signal.

To summarize, the proposed synaptic modification mechanism is consistent with several findings, but also requires further mechanisms. As such, the proposal outlined above constitutes a set of predictions regarding additional factors that should determine the sign and magnitude of synaptic modification.

Conclusions

The analysis and simulation results presented in this paper support the idea that the GeneRec family of learning algorithms are performing variations of error backpropagation in a recurrent network using locally available activation variables. However, there is no single GeneRec algorithm which is exactly equivalent to the Almeida-Pineda algorithm for backpropagation in recurrent networks since GeneRec requires symmetric weights yet it is not itself symmetry preserving.

The idea that the CHL algorithm is equivalent to a symmetry preserving version of GeneRec using the midpoint integration method is supported by the pattern of learning speed results for the different versions of GeneRec, and by the learning speed increases obtained when using the approximate midpoint integration method in backpropagation networks. Further, it was shown that CHL (and symmetric GeneRec without the midpoint method) can reliably learn the family trees problem, calling into question the idea that CHL is a fundamentally flawed learning algorithm for deterministic networks, as was argued by Galland (1993). Thus, the weight of the evidence suggests that CHL should be viewed as a variation of recurrent backpropagation, not as a poor approximation to the Boltzmann machine learning algorithm.

However, as a consequence of the differences between GeneRec and AP backpropagation (mainly the symmetry constraint), one can expect GeneRec to have somewhat different characteristics compared to standard backpropagation algorithms, and it may turn out that these differences have implications for psychological or computational models. Thus, the present analysis does not imply that just because there exists a biologically plausible form of backpropagation, all forms of backpropagation are now biologically plausible.

Finally, while CHL gave the best performance of the GeneRec networks in three out of the four tasks studied in this paper, the symmetry preservation constraint ended up being a liability in the 4-2-4 encoder task. Thus, the GeneRec-based derivation of CHL can have practical consequences in the selection of an appropriate algorithm for a given task. Also, this derivation allows one to derive CHL-like algorithms for different activation functions, and other network parameters.

Perhaps the most important contribution of this work is that it provides a unified computa-

tional approach to understanding how error-driven learning might occur in the brain. Given that the GeneRec learning rules are quite possibly the most simple and local way of performing a very general and powerful form of learning, it seems plausible that the brain would be using something like them. The specific biological mechanism proposed in this paper, which is consistent with several empirical findings, provides a starting point for exploring this hypothesis.

Appendix A: Plus Phase Approximation to Trial Step Activations

The relationship between the plus-phase activation of a GeneRec hidden unit j (h_j^+) and that which would result if an Euler weight update step were taken to reduce the error (denoted h_j^*) can be more formally established. This is done by simply re-computing the activation of the hidden unit based on the weights after they have been updated from the current error derivatives. Using the basic GeneRec algorithm with the difference of net-input terms instead of activation terms (this makes the computation easier), the trial step (starred) weights would be as follows:

$$w_{ij}^* = w_{ij} + \epsilon s_i(\eta_j^+ - \eta_j^-)\sigma'(\eta_j^-) \tag{2.29}$$

$$w_{kj}^* \approx w_{kj} + \epsilon o_k^-(\eta_j^+ - \eta_j^-)\sigma'(\eta_j^-) \tag{2.30}$$

Note that the *original* value of o_k^- is used here, whereas in the exact computation of the midpoint method in a recurrent network, the output activation value would change when the weights are changed. However, it is impossible to express in closed form what this value would be since it would result from a settling process in the recurrent network, so the original value is used as an approximation to the actual value. This is the only sense in which the following analysis is approximate.

The trial step weights above can then be used to compute the net-input that the unit would receive after such weight changes (denoted η_j^*) as follows (using the fact that $\eta_j^- = \sum_i s_i w_{ij} + \sum_k o_k^- w_{kj}$):

$$\begin{aligned}\eta_j^* &\approx \sum_i s_i w_{ij}^* + \sum_k o_k^- w_{kj}^* \\ &\approx \sum_i s_i w_{ij} + \sum_k o_k^- w_{kj} + (\eta_j^+ - \eta_j^-)\epsilon\sigma'(\eta_j^-)\left(\sum_i s_i^2 + \sum_k {o_k^-}^2\right)\end{aligned} \tag{2.31}$$

To simplify, let:

$$u = \epsilon\sigma'(\eta_j^-)\left(\sum_i s_i^2 + \sum_k {o_k^-}^2\right) \tag{2.32}$$

Which gives:

$$\begin{aligned}\eta_j^* &\approx \eta_j^+ u + \eta_j^-(1-u) \\ &= \eta_j^+ \text{ if } u = 1\end{aligned} \tag{2.33}$$

Thus, the plus-phase net-input (and therefore activation) is equivalent to a forward Euler step if the learning rate ϵ is set so as to meet the conditions in (2.33):

$$\epsilon = \frac{1}{\sigma'(\eta_j^-)\left(\sum_i s_i^2 + \sum_k {o_k^-}^2\right)} \tag{2.34}$$

Using a fixed learning rate which is smaller than that given by (2.34) would result in a starred (trial

step) activation value which is in the same direction as the plus-phase activation value (since the u term is bounded between zero and one) but not quite as different from the minus phase value.

Chapter 3

Theoretical Principles for Neural Learning

The development of an understanding of the theoretical principles underlying neural network learning and behavior has played an important role in advancing this field. These principles can provide the basis for deriving new learning algorithms, and they are important for understanding why certain algorithms tend to work better than others. This chapter provides a brief introduction to the theoretical framework of *statistical estimation*, which provides a coherent and useful way of viewing neural networks. Some basic findings within this framework suggest that there is a correspondence between how many constraints or prior assumptions are built into a network, and how easily the network will learn and how well it will generalize.

One can consider the role of associative learning and activity constraints which are part of the LEABRA learning algorithm as embodying abstract, principled assumptions about the nature of generally useful representations. There are a set of three such *representational principles* that are developed in this chapter, which form the theoretical basis of the LEABRA algorithm: *entropy reduction*, *information preservation*, and *standard representational form*. These principles are based on ideas that have been used to understand purely self-organizing algorithms, and thus are capable of developing useful representations even in the absence of error signals. However, a central hypothesis of this thesis is that error signals are needed in order to select and shape these representations so that they will be appropriate for a given task. In the simulations reported later in the thesis, it will be shown that, while performance is often surprisingly good in LEABRA without error-driven learning based on these representational principles, it is always better with error-driven learning.

In addition to the three principles regarding the development of useful representations, a set of principles regarding their implementation are developed. These *implementational principles*, the main one of which is also based on findings within the statistical estimation framework, play an important role in shaping the LEABRA algorithm. The main implementation principle is that of favoring *fixed* constraints over *adapting* ones. Further principles include the favoring of *robust* over

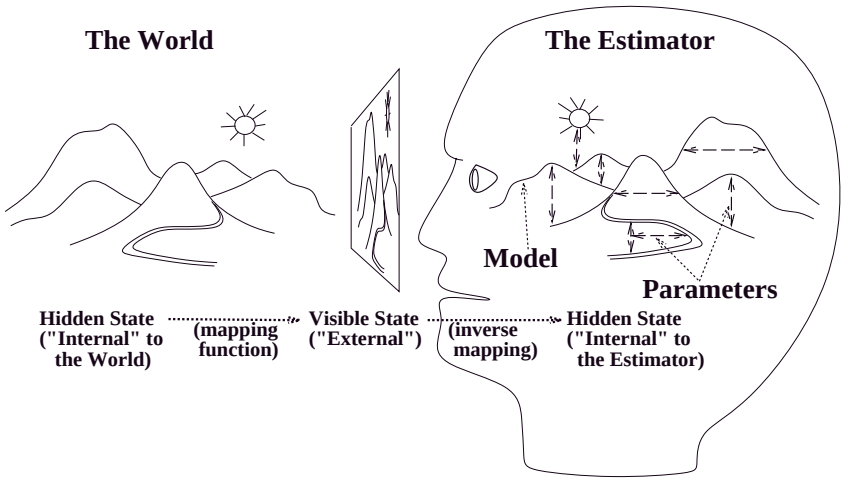

Figure 3.1: Schematic representation of the estimation problem.

fragile implementations, the use of *biological inspiration* where possible, and being careful not to violate *biological feasibility*. The impact of some of these principles is explored by comparing alternative formulations of LEABRA that violate them.

The Bias/Variance Dilemma and the Importance of Strong Models

Artificial neural networks can be thought of as a class of statistical estimation devices. Under this view, the goal of neural network learning is that of estimating or inferring the appropriate internal representations for the "structure" or regularities of the training environment, given some number of samples from that environment. One can divide the estimation problem into two components: the model and its parameters. Figure 3.1 shows a schematic representation of the estimation problem. As depicted, there is a world or environment (a.k.a. the *distal stimuli*) that is constituted by a particular set of states. These states are "hidden" because they are not directly known — instead, they must be estimated from the visible states of the environment (a.k.a. the *proximal stimuli*), which are available directly to the estimator. The job of the estimator is to use the sample of visible states to fit the parameters of an internal model of the world. This process can be thought of as learning the inverse of the mapping function that relates the hidden states of the world to the visible states (e.g., learning *inverse optics* to reconstruct the original 3D world from the 2D projection we receive on our retinas).

The distinction between the model and the parameters in the estimation process is like a nature-vs-nurture distinction. The model is whatever is assumed *a priori* by the estimator to be true about the world, and the parameters are whatever is adapted by experience. One of the principal dimensions along which estimators differ is the extent to which they rely on a strong model versus a clever parameter adaptation algorithm. For example, one could imagine having a very precise and accurate model of a particular environment, which would make it relatively easy to estimate because most

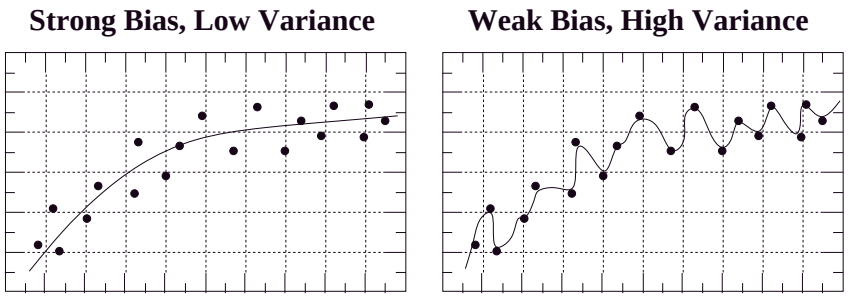

Figure 3.2: Schematic representation of the bias/variance dilemma, where the degree of bias is represented by the strength of a smoothness constraint in fitting a line to the set of points. If the "true" line is indeed smooth but the data is perturbed by independent random noise, a stronger bias gives a more consistent estimate across different samples of points, whereas a weak bias leads to high *variance* in the estimates because each new random sample of points will lead to a very different fit. However, if the "true" line is wavy, a stronger bias can lead to persistent *bias error* in the estimate.

of the work is done by the model. However, there are two problems with this. One is that such a model would be hard to come by, and the other is that it would be rather inflexible — the system would work well only with environments that happen to fit the model. Some traditional models of language acquisition have this character, in that they assume that people are born with a universal language, and learning is simply adapting some parameters of this language to fit the actual language one experiences (Chomsky, 1965).

Thus, systems with weaker, generalized models, more parameters and smart ways of adapting them, like neural networks, are attractive for their flexibility and ease of use (since not a lot of time needs to be spent coming up with a good *a priori* model). However, such systems have problems of their own: they require many more training examples because they have more parameters; they can have difficulty coming up with good estimates because they are not sufficiently constrained by the data contained in the visible states; and they tend to be more sensitive to noise in the visible states and therefore less capable of extracting the *structure* of the environment (i.e., the structure of the hidden states of the environment).

The weak model of purely error-driven neural networks is what leads to the problems with generalization and learning in deep networks as described in the introduction. Since the training task often does not impose enough constraints on the network, and the network does not have a strong set of constraints in its *a priori* model, the weights end up reflecting the residue of their random initial values as much as the structure of the task. This lack of structure causes problems, especially when the network is interactive, as it suffers from the butterfly effect, where small initial differences are magnified over setting through a turbulent activation space. Thus, it seems that there is room for improvement, especially in interactive networks but probably also in feedforward ones, by introducing appropriate constraints into the *a priori* model used by neural networks in learning.

Geman et al. (1992) explain the under-constrained nature of neural networks in terms of the *bias/variance dilemma*, which describes the tradeoff between strong *vs* weak *a priori* models (see Fig-

ure 3.2 for an illustration). *Bias* refers to the extent to which there is a strong *a priori* model "biasing" the estimation, and *variance* refers to the degree to which the same estimator will produce different estimates from different random samples of a given environment. According to this account, the way to minimize or avoid these problems is to introduce a stronger model into the estimator. Indeed, this is exactly the approach that is commonly taken, for example by using a form of *weight decay*, which addresses the variance problem at two levels. At the more general level, weight decay can be thought of as a more restrictive or precise *a priori* model which further constrains the kinds of solutions that the network can use (i.e., those with relatively few non-zero weights, Weigand et al., 1991). Also, it specifically reduces the number of parameters (weights) in the network, which results in fewer degrees of freedom in the estimator.

However, weight decay is a fairly "dumb" constraint, in that it is not itself capable of producing useful representations — a network using only weight decay will just end up with all zero weights. In this context, the idea behind LEABRA is that Hebbian associative learning can be thought of as a "smart" form of weight decay, in that it biases or constrains the network to produce representations of a certain form, but is also capable of generating useful representations on its own. Thus, LEABRA contains a stronger *a priori* model which includes constraints on the kinds of representations it will form, which are imposed by associative learning and other factors such as activity competition. However, it is important that this stronger *a priori* model in LEABRA does not throw out the essential flexibility of neural network learning with the "bathwater" of too many degrees of freedom in the estimator. It is important to realize that this is fundamentally a tradeoff, but nevertheless it may be possible (and indeed the performance of LEABRA indicates that it is) to constrain the model in a way that promotes the learning of a class of representations that are generally useful for a wide range of psychologically relevant tasks. The following section presents a set of principles which define the form of such representations.

Three General Principles Regarding the Form of Useful Representations

Within the framework of the statistical estimation paradigm described above, it seems somewhat implausible that there could be a way of *generically* constraining an estimator that would be useful to a wide range of problems. Indeed, it should be obvious that an *a priori* model that is not appropriate for a given problem will impair performance, not help it. Thus, if there are no constraints at all on the types of problems or environments the estimator will be exposed to, it is the case that all *a priori* models are equally useless (Wolpert, 1992). However, the actual world we live in (as opposed to the many hypothetical possible worlds an estimator might be exposed to) has particular properties that hold quite generally for many of the things that people actually learn about. These include basic physical properties like solidity, gravity, support, etc. which have been discussed in the developmental literature (Spelke, Breinlinger, Macomber, & Jacobson, 1992). However, they also include somewhat more abstract properties, like the fact that the visual world tends to be generally "smooth,"

in both space and time (Poggio, Torre, & Koch, 1985; Földiák, 1991; O'Reilly & Johnson, 1994). The principles described below are even more abstract than this, and constitute a very general claim about the nature of the world.

The literature on self-organizing neural network learning rules provides numerous examples of the application of heuristic *a priori* principles to extract generally useful information from an environment. Indeed, some believe that these self-organizing rules, implemented using Hebbian associative learning and other mechanisms, are sufficient to guide the development of most of the representations that underly human behavior. While the LEABRA algorithm specifically rejects this claim by using error-driven learning, the associative learning and other aspects of LEABRA are based on similar self-organizing principles. In these terms, the basic idea behind LEABRA is that these principles provide a means of usefully constraining the learning, but not fully determining it.

One example of a self-organizing principle is the *InfoMax* principle proposed by Linsker (1988), which holds that representations should maximize the amount of information they convey about the input signals they receive. The InfoMax principle can be instantiated in a neural network using relatively simple Hebbian associative learning rules, and gives rise to useful representations like the center-surround and oriented receptive fields found in the lower levels of the visual system. The full spectrum of heuristics underlying most self-organizing learning rules can be described according to the *minimum description length* (MDL) framework, which is based on two principles that might be characterized as *succinctness* and *accuracy* (Zemel, 1993; Rissanen, 1986). Basically, MDL states that the best way of encoding a set of data is with the simplest model possible which does not unduly sacrifice the accuracy of the data when encoded under this model. Thus, MDL can be thought of as a formalization of Ockham's razor favoring simple models over more complex ones.

Clearly, it is possible for succinctness and accuracy to tradeoff against each other, since the most accurate model is always a copy of the data itself, which is not very succinct, and many succinct models do not capture all of the data. Zemel (1993) argues that most self-organizing learning rules represent some kind of tradeoff between these two principles. For example, Linsker's (1988) InfoMax principle operates within a context which imposes a relatively fixed succinctness constraint, leaving accuracy (i.e., information) being the thing which is maximized. Given the generality of the MDL approach, the first two principles used in LEABRA correspond to particular ways of imposing succinctness and accuracy constraints. The third one represents a pragmatic way to avoid excess degrees of freedom in the estimation process.

Entropy Reduction: This is a particular form of a succinctness principle which is stated here in terms of the entropy of the state of a single hidden unit h as compared to that of the input signal X that this unit sees. This principle states that h should actively try to *reduce* the entropy of X. In one sense, this means that h should be a succinct representation of X. However, it goes further than that, since it is also desirable for h to be a succinct representation of only a *part* of X. Thus, implicit in this principle is the notion that there are a large number of different hidden units that participate in the representation of X, which allows each one of them to individually concentrate on only a part

of X. For example, a spatially invariant object representation reduces the entropy of a visual input signal by being equally active for all the different positions of a given object, and not active for other objects in any position. Thus, it is ignoring any location information in X, and even many variations in X that nonetheless correspond to the same object. We assume that there are other units that capture location information regardless of object identity, etc. Categorization and filtering are examples of entropy reduction: the category ignores all of the more specific information about the exemplar, and a filter is inactive for those inputs outside of its sensitivity range.

It is important to distinguish between the entropy reduction principle as stated here and Barlow's (1989) *minimum entropy* principle. The minimum entropy principle states that the units should develop representations such that they form a *factorial code*, where the summed entropy over all of hidden units would be at a minimum. Thus, Barlow's minimum entropy (and many other succinctness measures) is defined with respect to the aggregate representation over all the hidden units, whereas the entropy reduction principle as described here is explicitly focused on an individual unit. While the minimum entropy principle typically implies that each unit has roughly the same low probability of being active, which is consistent with the entropy reduction idea, another consequence of minimum entropy is that hidden units should be statistically independent of each other. No such restriction is implied by the entropy reduction principle as stated here. Entropy reduction is intentionally more vague about what units *should* encode in part because it is assumed that error-driven learning will be in effect, which will be responsible for determining many aspects of the representation. This principle merely states that there should be forces that shape the activities and weights of units which encourage them to narrow their focus on a dissociable part of the input signal, and develop categorical representations that apply to many different inputs.

It is important to be explicit about what this principle assumes about the nature of the learning environment. This principle assumes that the world (or at least that part of it of interest to the learner) consists of a number of dissociable entities, and that each of these entities can be usefully classified as belonging in more general categories. Both of these assumptions are relatively abstract and seem generally true of our world.

Information Preservation: This is a weak form of the accuracy principle which states that hidden units should preserve some information about the input signal X. Thus, once some of the "irrelevant" information has been eliminated by entropy reduction, it is important that the unit remain sensitive to some remaining distinctions between input patterns. In the case of invariant representations, it is important that they still provide information about *which object* is being viewed, even as they ignore *where* it is on the retina. Note that this is somewhat different, at least in emphasis, than the strong "reconstructionist" approach of the MDL framework as applied in Zemel (1993), where the objective is for the hidden unit representations to be able to fully reconstruct the input signal. Again, the assumption that error-driven learning is also taking place allows one to be more vague about the actual amount of information that should be preserved. For example, if there are possible distinctions between input signals which never end up being relevant for any tasks the system per-

forms, there is no reason to assume that the system should preserve this information. This principle simply assumes that some of the differences between entities in the world are relevant, and should be represented.

Standard Representational Form: This principle addresses the case where there are two hidden units that both have the same pattern of activity over the set of input signals, but with a sign change or some other "notational" difference between them. For example, when one unit is active, the other is inactive, and vice-versa. Since these two units would convey the same essential information about the input signal (i.e., the magnitude of their correlation term with the input would be the same), this principle states that this situation should be avoided by establishing standard forms of encoding information. The advantage of doing so is that the decision about what "form" to cast a given representation in represents an additional degree of freedom in the learning (estimation) process that could be eliminated without losing representational power. For example, the number of degrees of representational freedom would be cut in half if all sign-inverted versions of the same activity pattern were eliminated from consideration. In a neural network, it is possible to have a negative activation value multiplied by a negative weight, which has the same effect as a positive activation multiplied by a positive weight. The additional degrees of freedom implicit in this equivalency are unnecessary, and could be eliminated. Note that this is not so much an assumption about the world as it is a way of structuring the internal representations of the learning system.

As will be described in the next chapter, these constraints are implemented by the *MaxIn* associative learning rule and *ReBel soft k-winners-take-all* competitive activation function used in LEABRA. The activation function limits the number of active units in a graded manner (hence the term *soft*), which imposes an entropy reduction constraint, and also establishes a standard representational form. The learning rule imposes both an entropy reduction and information preservation constraint in an adaptive, balanced manner.

Implementational Principles

The above discussion of representational principles says nothing about how these ideas should be implemented in actual neural network algorithms. There are many degrees of freedom in the implementational process, and there might be important differences in the resulting performance that depend on implementational choices. Thus, the following set of implementation principles are proposed to help constrain these choices. Further, the first and perhaps most important principle will be substantiated by comparing alternative implementations that violate it.

Fixed *vs* Adapting Constraints: One important way of carving up the space of possible implementations of any kind of principle or constraint in a neural network is along the fixed *vs* adapting dimension. A *fixed* constraint is one that is built into the network as part of the architecture, is not subject to learning, and cannot be overridden by other constraints. Thus, it can truly be considered as a part of the *a priori* model, and is likely to have the beneficial effects of more strongly biasing the

estimator, as described above. An *adapting* constraint, on the other hand, is one that is added into the learning algorithm in order to introduce a pressure on the system to develop in a certain way, but it can be contradicted by other pressures in the learning rule. An adapting constraint typically increases the number of parameters that are being learned, and can thus actually *increase* the variance problems associated with the estimator.

Weight decay is a good example of an adapting constraint, which is consistent with the fact that it typically takes *longer* to train a network with weight decay than without. Thus, even though weight decay can be thought of as providing a more constrained *a priori* model, the way this constraint is implemented can have a large impact on how much the advantages of this more constrained model are actually exhibited. Consider, for example, a constraint similar in some ways to weight decay, where only a fraction of the connections between units are created in the first place. This is a fixed constraint, and networks trained with it will typically learn faster than a network with weight decay, if they are capable of learning the problem at all. However, because this constraint is not subject to modification, it can be inappropriate for a given problem, and cause the network to be unable to learn at all. Thus, the optimal implementation of a given constraint is probably an empirical issue that can only be resolved by evaluating the performance of various possibilities.

Some of this empirical investigation is undertaken in this thesis with respect to the different ways of implementing the three representational principles described above. The optimal version represented by LEABRA uses fixed constraints to implement entropy reduction and standard representational form, but adapting constraints to implement additional entropy reduction and information preservation. This version is compared to one where everything is done with adapting constraints.

Robust *vs* Fragile Mechanisms: There are often fragile and more robust ways of implementing a given constraint. In general, the robust implementation is favored in the development of the LEABRA algorithm. For example, an activity constraint could be implemented by setting a relatively high threshold (or a large negative bias) on units. However, this is not robust to overall differences in levels of activation, and requires precise tuning of the threshold parameter for a given problem. A more robust version would automatically set the threshold according to some measure of activity level in the units. As in this example, it is often the case that a very simple implementation is fragile, whereas it takes a slightly more complicated mechanism to make it robust. As another example of this, an activity constraint that is imposed by direct lateral inhibition between units, which seems simple and straightforward, turns out to be highly unstable except when only a single unit is allowed to be active. Thus, a slightly more complicated but much more robust implementation via the ReBel *soft k-winners-take-all* function is used in LEABRA.

Biological Inspiration: Wherever possible, the known properties of the cortical biology are evaluated in terms of their relationship to a given principle. For example, a prominent feature of cortical circuitry is recurrent inhibitory feedback from interneurons. While conclusive evidence is not available, learning has not been demonstrated in these inhibitory synapses, suggesting that they might implement a relatively fixed form of a constraint on the activity of excitatory neurons. Thus, it is

tempting to interpret this as evidence that the brain uses a fixed activity constraint, which would be one way of implementing the entropy reduction principle. Given that this biological property is synergistic with the computational principles of entropy reduction and fixed constraints, it is therefore all the more desirable to include it in the algorithm. Similar arguments can be made about biological properties like recurrent connectivity, and the fact that most neural communication in the cortex appears to be excitatory, as will be discussed later.

Biological Feasibility: Cases where it is highly unlikely that the biology could implement a given property of the algorithm are avoided. Thus, explicit error backpropagation is avoided in favor of a version of error-driven learning that can use locally available, activation-based signals, which is much more biologically feasible. This is different than biological inspiration, since error-driven learning in this manner is certainly not a prominent known feature of the cortical biology. This constraint is not a very strong one, though, because aside from glaring problems like non-locality or contradiction of well-established facts, it is often impossible to argue that one implementation is more biologically feasible than another.

Chapter 4

Implementing the Principles in Self-Organizing Mechanisms

The preceding chapter provides the theoretical basis for the implementation described in this chapter of the two self-organizing components of the LEABRA algorithm. The first component is based on a mathematical framework called *relative belief* (*ReBel*), which is used to derive a network architecture and activation function that implement a fixed form of the entropy reduction principle based on activation constraints, and establish a standard representational form. The second component is a self-organizing, Hebbian associative learning rule called *MaxIn*, which is derived using the ReBel framework, and implements both entropy reduction and information preservation in an adaptive manner. The combination of both the activity constraints and the associative learning rule constitute the implementation of the *a priori* model based on the representational principles described in the previous chapter. In LEABRA, these are assumed to work in conjunction with a third component which is the biologically feasible GeneRec error-driven learning algorithm described in Chapter 2.

The next section of this chapter describes a particularly effective way of combining error-driven and self-organizing learning, which provides an error-driven component to the learning of *self-structure* representations of the input and output patterns. In a standard error-driven learning framework, the error signals are based, in psychological terms, on the difference between an *expectation* generated in response to an input, and the actual *outcome* or output. The MaxIn self-organizing learning in LEABRA causes the network to represent the correlational self-structure of the individual patterns (input and output) themselves (in addition to the correlational structure in the input-output mapping). For the same reasons that both error-driven and associative learning work better for learning input-output mappings, the combination of these two should work better for representing the individual self-structure of the input and output patterns. Thus, a mechanism for introducing error signals into the development of these representations is needed. Such a mechanism, called the *auto-encoder* (AE) version of LEABRA, is described in the third section of this chapter.

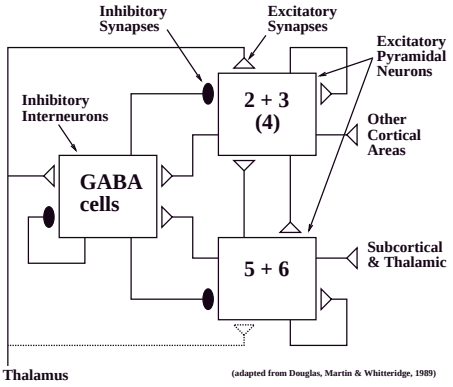

Figure 4.1: Canonical circuit of the neocortex, adapted from Douglas et al. (1989). Thalamic input comes primarily to layer 4, but also to other layers. Excitatory neurons in the superficial layers (2 + 3) project to other cortical areas, and locally to the deep layers (5 + 6) and to themselves. All excitatory inputs and outputs also contact the GABA cells, which provide inhibition to the excitatory neurons, and to themselves. Excitatory neurons in the deep layers project to subcortical and thalamic areas. This circuit describes the basic properties common to all cortical areas.

Architecture and Neural Activation: The Relative Belief Framework

The foundation of any learning algorithm is in its architecture and the way that units communicate and relate to each other within the network. This architectural foundation defines the signals on which learning operates. As described in the previous chapter, one of the central ideas behind LEABRA is that as many constraints as possible should be built into the architecture as *a priori* biases. This constrained architecture will reduce the size of the space that learning has to explore, and potentially provide richer signals to inform the direction that learning should take.

Since there is a considerable amount of data available regarding the basic structure and circuitry of the neocortex, the implementational principle of biological inspiration can be taken seriously in considering the basic properties of neural architecture and activation that should be incorporated into LEABRA. A brief overview of this data is presented below, together with the interpretation of it used in the ReBel activation function. This is followed by a discussion of some important computational issues for the implementation of activity constraints. Finally, the ReBel function is presented in detail.

Inspiration from the Neocortex

It is possible to provide a canonical circuit diagram for cortical connectivity that describes the basic connectivity common to all cortical areas (Creutzfeldt, 1977; Szentágothai, 1978; Shepherd, 1988; Douglas et al., 1989, see Figure 4.1). The excitatory and inhibitory connectivity in the neocortex is stereotyped and highly regular, with excitatory connections existing both within and between areas of cortex, but inhibitory connections occurring only within relatively small localized regions. The principal excitatory cells (pyramidal neurons) project only excitatory neurotransmitter (glutamate) onto both local inhibitory interneurons, and onto other pyramidal neurons, both in other areas

and in typically well defined patterns within the same cortical area. The GABAergic inhibitory interneurons, which also receive excitatory connections from the external projections that synapse onto the principal neurons in the local region, have inhibitory synapses on excitatory cells that surround them. Both this inhibitory circuitry and restrictions on the sign of information signals in neocortex can be related to the representational principles of entropy reduction and standard representational form, and to the critical properties of the ReBel activation function.

Entropy Reduction Through Inhibitory Circuitry

The activation constraints imposed by the ReBel activation function, while functionally motivated, take a form which is based on an interpretation of the role of inhibitory interneuron circuitry in cortical systems (including the three-layered cortex of the hippocampal formation). Many researchers have viewed the inhibitory circuitry as providing a negative feedback system that effectively normalizes or controls the overall level of activity of the principal excitatory neurons. Thus, as more neurons get more active, they also activate the inhibitory interneurons more, resulting in greater inhibition, which checks the growth of activity. This activity regulation is especially important given the positive feedback loops that exist between recurrently connected principal neurons, which can lead to an unstable cascade of activity if not checked — witness the phenomenon of epilepsy, which occurs when the inhibition is insufficient to regulate the excitatory activity.

The activity regulation imposed by the inhibitory circuitry can be thought of as an implementation of the entropy reduction principle, in that neurons in a region receive inputs from many areas, but reduce this to a representation having a limited number of active neurons. Computationally, this inhibition has the effect of inducing a competition between neurons, a property which has been shown in many self-organizing neural network models to result in units that specialize on representing different aspects of the input, which is also consistent with the entropy reduction principle.

The effect of feedback inhibition can be idealized in terms of a *k-winners-take-all* (*kWTA*) function, which corresponds to a negative feedback system with a fixed stable point that is the same (i.e., approximately k units active out of N total at any given time) regardless of the total amount of excitation coming into a given layer (Gibson, Robinson, & Bennett, 1991; O'Reilly & McClelland, 1994; McNaughton & Morris, 1987; Torioka, 1979). In effect, the inhibition provides a *floating threshold* for activity based on overall levels of activity in the layer. While the kWTA form of this threshold is clearly an idealization of the actual dynamics of the inhibitory system, it is a useful one that is probably not too far from the truth. The ReBel activation function imposes a *soft* kWTA constraint on the activations of units within a layer, which is soft in the sense that the constraint is imposed in a *graded* (as opposed to binary) manner. As such, it corresponds to one way of generalizing the commonly-used *soft WTA* or *SoftMax* activation function to the case where multiple units (instead of only one) are allowed to be active at the same time.

While the mathematical framework used to develop the ReBel function is based on Bayesian hypothesis-testing ideas and is somewhat removed from a plausible biological implementation, there

is nevertheless a clear distinction in the two terms that make up the ReBel equation between the physiological properties assumed of individual neurons, and those of the inhibitory interneuron circuitry of the cortex. The individual neuron is assumed to have a propensity for activity which is a simple sigmoidal nonlinear function of the input to the neuron. However, this individual propensity for activity is modulated (multiplied) by a function of the ratio of the individual activity over that of other neurons in the layer as reflected in the inhibitory inputs coming from the inhibitory interneurons. It is this latter term which implements the kWTA constraint, which acts much like shunting inhibition on the firing rate of excitatory neurons. Thus, ReBel can be thought of as a relatively simple analytical approach to understanding the role of the inhibitory interneurons in controlling the activity levels of principal excitatory neurons. Future research will be focused on determining how well this simplification approximates a more realistic form of soft kWTA behavior based on simulated inhibitory interneuron circuitry.

Standard Representational Form Through Sign Constraints

The constraints on the signs of the activations and weights of pyramidal neocortical neurons can be thought of as implementing a standard representational form, as follows:

Positive-only Activations: Information is transmitted by neurons only when they are active. Thus, unlike some artificial neural network units, which can take on negative and positive activation states, real neurons can only emit signals ranging from 0 (no spikes) up to some positive number of spikes per second (with a maximum of around 200 or so). This constraint makes sense in terms of the standard representational form principle, since representations are forced to use positive activation as a signal, whereas (-1..1) units can use both negative and positive activations as a signal.

Positive-only Weights: Representational neurons (pyramidal neurons) only communicate directly via excitatory weights. In combination with positive-only activations, this uniquely constrains the form of signals that neurons can use, which is obviously in accord with the standard representational form principle. Thus, in a system with positive-only activations and weights, a given input-output mapping (in one layer) can be implemented with only one set of weights (assuming the input/output patterns are non-redundant). In contrast, a system that allowed either negative weights or activations could implement the same input-output mapping in different ways by variously canceling out the influences of pairs of units which contribute positive and negative inputs.

While it could be argued that the local-circuit inhibitory neurons are sending specific inhibitory signals, this may not be a very plausible hypothesis, since these neurons tend to interact with a large number of local excitatory neurons using GABAergic synapses, which have not been shown to be capable of associative, activity-based modification. Thus, whatever information they convey is likely applied without learning-driven specificity to a large number of neurons. In addition, the firing specificity of the inhibitory interneurons in electrophysiological recordings is very poor — they tend to fire at a relatively constant, high rate. These properties are consistent with the role of these interneurons as providing a relatively fixed activity constraint, and not with a role in representing specific,

learned information.

To summarize, the picture that emerges from consideration of these general properties of the neocortex is that the brain has several fixed constraints which are consistent with the entropy reduction and standard representational form principles. These fixed constraints can be directly incorporated into LEABRA via the ReBel activation function, resulting in both increased biological relevance and the computational advantages associated with these constraints.

Computational Issues in Activity Regulation

While the fixed constraints suggested by the biology on the sign of activations and weights are fairly unambiguous in their meaning and implementation, the issue of activity regulation and its computational effects is considerably more complex. To explore the space of possible activity constraints, one can begin by examining the extreme cases. At one extreme, a standard backprop network has no activity constraints, and has completely distributed representations. In such networks, it is useful to think of the units as each representing a sub-dimension of the input space, with the activation of the unit signaling the value of a given input pattern along this dimension. Thus, the hidden units provide a set of useful "basis vectors" for representing the input, and a large number of such units cooperate to represent a given input pattern. An important advantage of distributed representations is this ability to represent instances by a combination of relevant features, which facilitates many kinds of neural processing, including pattern completion, generalization, similarity-based processing, etc. (Hinton, McClelland, & Rumelhart, 1986; Seidenberg & McClelland, 1989). For these reasons, the ability to learn and use distributed representations has been identified in this thesis as one of three critical functional properties for a model of neocortical learning.

However, it seems that distributed representations can be taken too far. An example of this was discussed in the introduction, where the problematic consequences of unconstrained distributed representations were reviewed. Essentially, without a pressure to specialize, units can have quite random-looking weight vectors and still enable the problem to be solved. Thus, one way of avoiding this problem is to employ a form of entropy reduction via activity regulation, which would have the effect of forcing units to be more selective in what they represent, and hopefully therefore more likely to pick up on important structure in the environment.

In addition to the motivation for activity constraints based on entropy reduction, there is an important practical motivation for activity constraints in systems employing Hebbian associative learning. It is commonly known that Hebbian learning, since it is essentially a positive feedback system, suffers from a "rich-get-richer" problem that can lead to one or a few units taking over the entire representational space. Activation-based competition (e.g., the prototypical *competitive learning* algorithm described by Rumelhart & Zipser, 1986) is one way in which this problem is dealt with, since it forces some units to take responsibility for the current input pattern and others not to. In the larger context of the main goal of the thesis, which is to explore combined associative and error driven learning systems, this means that some kind of activity constraint is necessary for such an

endeavor.

The extreme form of activity regulation is a *winner-take-all* (WTA) activity constraint, where a single unit represents an entire input pattern. This kind of localist representation obviously lacks the crucial advantage of distributed representations — the ability to represent instances by a combination of relevant features. Despite the clear advantages of distributed representations, WTA functions have been widely used in the self-organizing literature. Aside from the conceptual and practical simplicity of the WTA approach, it also has a particularly useful statistical interpretation in terms of a Bayesian analysis of unit representations (Nowlan, 1990), and for the gating network in the *mixture of experts* framework (Jacobs, Jordan, Nowlan, & Hinton, 1991). This analysis makes it possible to understand the role the activity constraint is playing at a mathematical level, and it can be used to derive learning algorithms that are consistent with the WTA model.

It seems reasonable that the intermediate case between a fully distributed and a WTA representation, a *sparse distributed representation* (e.g., the ReBel kWTA function), might represent the best tradeoff between the advantages of a distributed representation and the benefits of activity regulation (see Dayan & Zemel, 1995; Zemel, 1993, for similar arguments). In this case, only a subset of the units are allowed to be active at any given time, which introduces both cooperation and competition into the representation — the active units have to cooperate to represent the current input pattern, but they also have to compete with the other units to get active in the first place. This should result in each unit picking up on significant regularities in the input/output space, resulting in a distributed representation that is also more systematic. While Zemel (1993) associates a sparse distributed representation with Barlow's (1989) minimum entropy principle, this tends to emphasize the redundancy elimination or competitive aspects of the representation over the cooperative aspects. Indeed, one advantage of a distributed representation over a localist one is that it provides useful similarity structure for subsequent processing by deeper hidden layers — redundancy elimination attempts to remove this structure by making each unit statistically independent. Thus, there must be a tradeoff between competition and cooperation.

Unfortunately, it can be difficult to develop a formal treatment like that of the WTA function for the intermediate case between the fully distributed and WTA models. In the WTA case, all units are treated as mutually exclusive and exhaustive, which is easily formalized, while in the fully distributed case, they are treated as independent, meaning that the aggregate probability of a pattern is just the product of the individual unit probabilities, which is also easy to formalize. However, in a sparse distributed or kWTA system, there is a dependency between the units which can be hard to express without having to evaluate probabilities over all possible combinations of active units, which quickly becomes impossible in even moderately-sized networks (not to mention the problems with biological plausibility of this kind of computation). Nevertheless, there are several classes of approaches that have been developed formally. Perhaps the simplest of these is to continue to treat each unit individually, but to do so in a way that makes the activity of each unit relative either to a fixed threshold or other pre-specified probability distribution (e.g., Zemel, 1993) or to a function

computed over other units in the same pool. A particular version of this latter approach is what is used in the ReBel function.

Other approaches to formalizing sparse distributed representations involve using the assumption that each bit in the output representation is driven by a single underlying cause (i.e., unit) in the hidden layer to constrain activity levels (Dayan & Zemel, 1995), or a *noisy-or* function that accomplishes roughly the same thing (Saund, 1994, 1995). In some sense, this is just another way of implementing a WTA constraint, but it offers more flexibility than a strict WTA in that different hidden units can simultaneously be active as long as they are responsible for different output bits. Finally, more complicated hierarchical systems have been used to get around the WTA assumption. For example, Jordan and Jacobs (1994) have developed a hierarchical version of the mixtures of experts framework, and Dayan, Hinton, Neal, and Zemel (1995) have developed a hierarchical model where activities in the previous layer are constrained by *generative models* in subsequent hidden layers. This latter form of constraint is very general in that it can be anything a set of hidden units can encode. However, all units in each layer are treated as conditionally independent to simplify the mathematics, so it might be that these networks will suffer from being underconstrained in much the same way as a backprop network.

Relative Belief (ReBel) Analytical Framework

With the issues discussed above in mind, the key design criteria for the ReBel activation function are as follows:

- Total activation over a layer should have a relatively fixed kWTA upper bound (i.e., no more than k active units). A fixed maximum is needed to provide a strong impetus for the specialization of representations (entropy reduction), and to control the associative learning positive feedback syndrome.
- Flexibility in total activation should exist in the range between 0 and k active units, depending on how the units learn to carve up the representational space. This allows representations to *not* participate in a given form of processing when they are not relevant.
- Unit activation should be graded, and reflect both the *individual* level of support a unit receives from its inputs, as well as the strength of a *relative* comparison to other units in the layer.

These criteria can be met with a function based on a simple characterization of the canonical cortical circuit described previously. In particular, one can separate the activity of a neuron into two components: 1) The individual propensity for activity based on levels of excitatory input from other cortical neurons or thalamus; 2) The comparative propensity for activity relative to the level of inhibitory input, which is a function of the activity of other neurons in the layer. In the ReBel model, these two terms are represented by probability functions, and have a multiplicative relationship. Thus, the overall probability of firing is the combination of two independent factors represent-

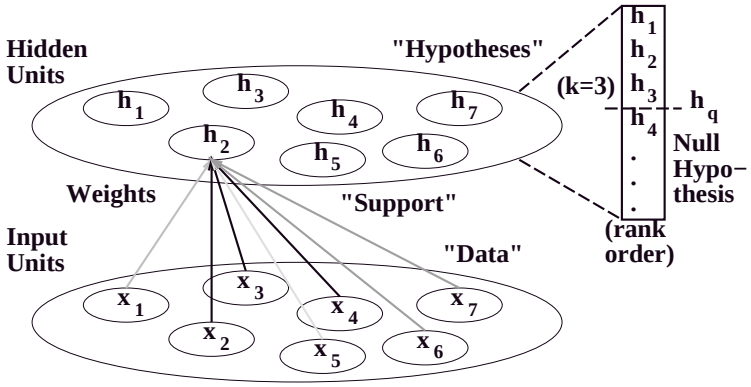

Figure 4.2: Hypothesis-testing framework for unit activations in ReBel. Units represent hypotheses defined by their weights to input units. The overall likelihood of a hypothesis is a function both of the absolute level of support and the relative level of support compared to other hypotheses in the layer, which is computed by comparison to a null hypothesis h_q, which represents the probability of a middling hypothesis.

ing the individual and relative probability of firing. Further, the relative term based on inhibitory input is defined so as to be greater than .5 for the k units with the greatest individual probabilities for activity, and less than .5 for the remainder, establishing the fixed kWTA upper bound on activity. The multiplicative relationship of the relative term with the individual probability allows flexibility between 0 and k units because the individual probability of a "winning" unit may nonetheless be rather low.

The formal derivation of ReBel is based on Bayesian hypothesis testing as diagrammed in Figure 4.2. Hidden units are considered to represent hypotheses h_j, as parameterized by their weights from input units, which represent the data $\mathbf{x}_p$ of pattern p out of environment X (or the belief values of other hidden units, in multi-layer networks). The activation value of a hidden unit represents the graded "belief" in the truth-value of its hypothesis, as evaluated by using equations for manipulating probabilities in validity-preserving ways. However, as is often the case in the Bayesian framework, these probabilities are better thought of in terms of the probability that a "reasonable agent" (defined as follows) would believe the hypothesis to be true, rather than in terms of observable frequencies of events occurring in the world.

The appropriate interpretation of the graded belief values used in ReBel can be clarified somewhat by considering an alternative interpretation to the one just provided. In this interpretation, the belief value represents an estimate of the degree to which the hypothesis is true *in the real world*, as opposed to just the extent to which it is believed to be true by the network. In this alternative case, we specifically assume that things in the world have graded (continuous) states of truth-value, and that our network is trying to accurately represent these graded values in the activations of the units in the network. An example of this can be considered in the case of representing the orientation of lines. If the unit is representing the hypothesis "the input contains a vertical line," and a not-quite-vertical line is presented in the input, one might expect the unit to produce a graded belief value that reflects the extent to which the line is not quite vertical (e.g., .85). Under the alternative interpretation, we

would consider this to represent an underlying not-quite-verticality of the line in the real world.

In contrast, the actual usage of belief values in ReBel is somewhat different in an important and potentially subtle way. In order to make this clear, we need to distinguish between the *true (distal) states* of the world and the *apparent (proximal) states* (a.k.a. evidence) of the world, which is all that we (the network) have available to evaluate the true states of the world. ReBel assumes that the true states can be cast as binary-valued entities (i.e., the line is either vertical or it is not vertical), and further that the apparent states are noisy, transformed, etc. reflections of the true states (e.g., there are distortions due to optics, atmospheric distortion, retinal coding, etc.).

The belief in the truth value of any hypothesis is based on the fit between the apparent states and what we imagine the apparent states generated by the true states would be. This fit, called the *likelihood* can be written in Bayesian terminology as $P(\mathbf{x}_p|h_j)$. In other words, we expect certain forms of evidence in the case of a particular hypothesis, and evaluate the truth value on the match between this expected evidence and the actual evidence (apparent states). Due to the noise, etc, this match is rarely going to be perfect, but some fits are better than others. If we imagine the noise to be random (not systematically biased in favor of any specific hypothesis), then it is improbable that a good fit is due to noise, and in general, the extent to which the evidence fits our expectations should make us more confident that our hypothesis is true. Thus, we can use the goodness-of-fit of the evidence as a way of expressing the our degree-of-belief in the hypothesized true state of the world. Of course, when comparing different hypotheses, we need to take into account the prior probabilities of these different hypotheses, which is what effectively enters into the evaluation of $P(h_j|\mathbf{x}_p)$ as a function of the likelihood $P(\mathbf{x}_p|h_j)$.

Thus, I can have a .85 degree of belief that a given line is vertical based on a visual stimulus which might (in the true state of the world) be not quite vertical. This .85 does *not* reflect the *certain* belief (knowledge) that the line is actually .85 off of vertical (defined in some as-yet-unspecified way). This interpretation of belief belies both the presence of noise, and perhaps more importantly for the ReBel perspective, the *context* in which the hypothesis is being evaluated. Thus, the .85 value reflects that, given some level of *uncertainty* due to noise, etc, my *subjective* belief in the verticality of the line given the current evidence is .85. As a line gets more vertical, the evidence becomes more consistent with that expected of a vertical line, and the belief value typically increases. In ReBel, the actual values of belief vary with the nature of the space of hypotheses being entertained. For example, one could have a large number of very narrowly tuned detectors representing hypotheses of lines being at very small increments (which could be taken to the continuous limit), or one could have a fewer broadly tuned detectors representing hypotheses of lines being at 45 degrees (for example). In the former case, the expected level of uncertainty regarding the fit between the evidence and the true state may be sufficiently low that it makes sense to have a fine-grained set of hypotheses. In the latter case, the uncertainty may be greater, so that it is not useful to consider that fine-grained of hypotheses.

In neural network terms, this example is reminiscent of the coarse vs. localist coding issue. As such, the expected level of uncertainty is only one factor that might influence the spacing and tuning

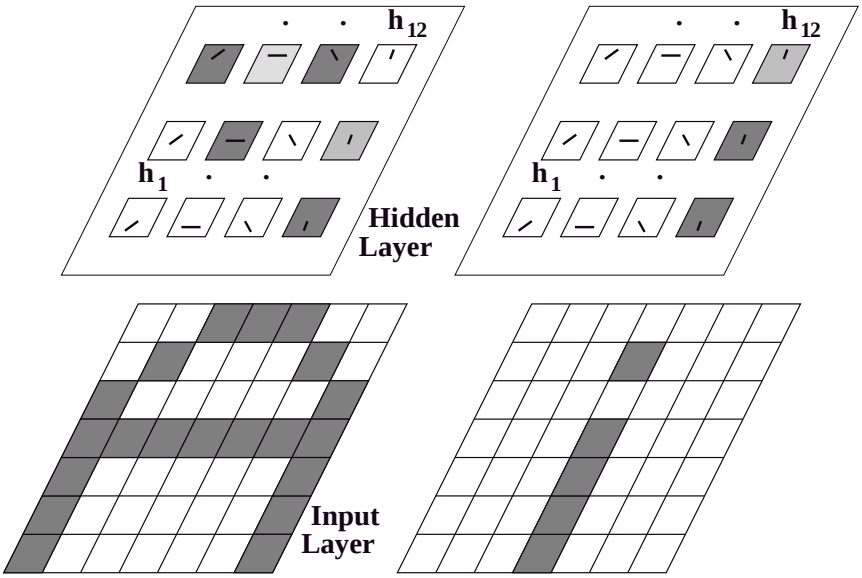

Figure 4.3: Example of a simple 2-layer network for representing the features of digits. The input layer represents the image of a digit, and the hidden layer contains units that represent the presence of oriented lines in each of 3 different regions of the input. Thus, h_1 is the hypothesis that "there is a 45 degree line in the lower third of the input", which is not very probable in the given example. There is a kWTA constraint of 4 active units over the whole hidden layer, so that with the 'A' input, the relative component of ReBel lowers the activity of two of the hidden units, while the 'i' input activates so few hidden units that the individual component of ReBel results in less than 4 total units active.

of hypotheses — issues of efficiency, relevance to environmentally important distinctions, etc. are also critical. In ReBel, the weights are what determine the nature of the hypothesis, which, given that the weights have exact values at specific points in time, correspond to a specific (i.e., binary) hypothesis about the true state of the world. The ReBel function then provides a means of evaluating the evidence for this hypothesis as a function of the fit between these weights and the sending unit activations, which provide the evidence. Built into this process is an expectation of uncertainty (i.e., in the evidence about the true state of the world provided by the sending unit activations, and in the specific weight values). The values of the weights, and the belief states of other hypotheses, can both influence the effective level of this uncertainty. Thus, it is the role of the learning to determine these parameters by adapting the weights.

In order to provide a specific example of how ReBel works, consider the simple digit-recognition network shown in Figure 4.3. The hidden layer contains units which represent hypotheses such as "there is a 45 degree line in the lower third of the input," and multiple such hypotheses can be true of a given input image. The two example inputs illustrate the contributions of the individual and relative (kWTA) components of the ReBel function to the overall belief (activation) value of the hidden units. Assume that there is a kWTA constraint such that only 4 hidden units can be active. Thus, when the 'A' input is presented, the relative component of ReBel, which implements this kWTA constraint, causes two units which would have a reasonable level of individual support from the input to have a

lower (below .5) overall belief value. In contrast, the 'i' input results in significant individual levels of belief for only 3 units, so that this component of the ReBel function causes the resulting belief level to be low for the top unit, practically non-existent for any fourth potential unit. Thus, the overall activity level ends up being below the maximum of 4 — this is an important property of the ReBel function.

The two components of ReBel (individual and relative) amount to two different ways of evaluating the truth-value of the unit's hypothesis, which can be thought of as two variants of the basic hypothesis represented by the unit. The individual evaluation of the truth-value of h_j, denoted h_j^i, considers whether the data indicates the hypothesis is true ($h_j^i = TRUE$ or simply h_j^i) compared only to the possibility that it is not true ($h_j^i = FALSE$ or simply $\overline{h_j^i}$), without consideration for the space of other hypothesis which might also account for the data. Thus, using the above example, the evaluation of this hypothesis for data indicating a line at a 45 degree angle would, regardless of the status of other hidden units, always result in a high truth value (near 1), while a -45 degree angle would result in a low truth value (near 0). Intermediate angles values would presumably produce intermediate truth values (e.g., a 0 degree angle would result in a .5 individual truth value).

In contrast, the relative evaluation of the truth-value of h_j, denoted h_j^r, considers whether the data indicates the hypothesis represented by this unit is true ($h_j^r = TRUE$ or simply h_j^r) compared relative to a *null hypothesis*, h_q^r. This null hypothesis is computed as a function of the other hypotheses in the layer, which is what makes it a relative evaluation of the given hypothesis, and it ends up acting like a floating threshold of inhibition for the kWTA function. Thus, using the previous example, the null hypothesis for the 'A' input might actually be quite strong, since there are more units that could be activated than the 4 allowed. Thus, only those 4 that are very strongly supported by the input will end up being more probable than this strong null hypothesis. In contrast, the 'i' input results in a relatively weak null hypothesis.

It is the use of this null hypothesis instead of an explicit consideration of all possible combinations of the other hypotheses that makes this approach analytically tractable. However, this also means that the relative evaluation is only as meaningful as the null hypothesis, since it does not provide any objectively meaningful probability value as a function of the combination of other hypotheses in the layer. This tradeoff is justifiable given the intractability of the fully combinatorial case, together with a bias towards thinking that the brain is likely to use reasonable shortcuts.

Having defined all of the critical terms, it is now possible to express the overall ReBel function:

$$ReBel(h_j, \mathbf{x}_p) \equiv P(h_j^i, h_j^r | \mathbf{x}_p) \tag{4.1}$$

which basically says that the belief in the truth value of the hypothesis represented by unit j is equal to the probability of the two variants of the hypothesis (i.e., the two ways of evaluating it) given the current input pattern. If we assume that these two ways of evaluating the hypothesis are conditionally independent of each other (which is probably at least approximately true), then the ReBel function

can be written in terms of the product of two terms, one individual and one relative, which is exactly what is desired:

$$ReBel(h_j, \mathbf{x}_p) = P(h_j^i|\mathbf{x}_p)P(h_j^r|\mathbf{x}_p) \tag{4.2}$$

The following two sections address the definition and computation of each of these two terms. Both sections rely on the same Bayesian mathematics, which is reviewed here first. In both ways of evaluating the hypothesis, there are two mutually-exclusive, exhaustive hypotheses that are considered. In the individual case, one hypothesis is that h_j^i is true, and the other is that it is false. In the relative case, one hypothesis is that h_j^r is true, and the other is that h_q^r is true. We will label the first of these hypotheses p and the other one q. In each case, we want to evaluate $P(p|x)$, which can be expressed using Bayes' formula:

$$P(p|x) = \frac{P(x|p)P(p)}{P(x)} \tag{4.3}$$

Because we are considering only two mutually exclusive, exhaustive hypotheses (p and q), the probability $P(x)$ can be written in terms of its probability under each of these hypotheses (e.g., $P(x|p)P(p) + P(x|q)P(q)$), which results in the following simplification of (4.3):

$$\begin{aligned} P(p|x) &= \frac{P(x|p)P(p)}{P(x|p)P(p) + P(x|q)P(q)} \\ &= \frac{1}{1 + \frac{P(x|q)P(q)}{P(x|p)P(p)}} \\ &= \frac{1}{1 + \left(\frac{P(x|p)P(p)}{P(x|q)P(q)}\right)^{-1}} \end{aligned} \tag{4.4}$$

This expression is particularly interesting, because it makes an important link with the *odds ratio* of p over q, which is:

$$\begin{aligned} O(p,q) &= \frac{P(p|x)}{P(q|x)} \\ &= \frac{P(x|p)P(p)}{P(x|q)P(q)} \end{aligned} \tag{4.5}$$

Odds ratios (or more commonly, the natural log of the odds ratio) provide the starting point for a wide variety of applications relevant to hypothesis testing and belief assessment (Pearl, 1988; Kass & Raftery, 1993; Anderson, 1990). The link between (4.4) and the log of the odds ratio comes via what might be called a "normalization" of the log odds onto the $(0..1)$ scale appropriate for probabilities. Since the log odds gives a value that can range over the real values, a squashing function like the logistic can be used to normalize it. Indeed, the application of the sigmoidal logistic function:

$$Sig(x) = \frac{1}{1 + e^{-\gamma x}} \tag{4.6}$$

to the log odds ratio of p over q gives an expression nearly identical to (4.4):

$$\begin{aligned} Sig\left(\log O(p,q)\right) &= \frac{1}{1+e^{-\gamma \log \frac{P(x|p)P(p)}{P(x|q)P(q)}}} \\ &= \frac{1}{1+\left(\frac{P(x|p)P(p)}{P(x|q)P(q)}\right)^{-\gamma}} \end{aligned} \tag{4.7}$$

The only difference is the presence of the gain parameter γ that determines the "sharpness" of the logistic function.

The following table, adapted from Kass and Raftery (1993), compares standard heuristic evaluations of the log of the odds ratio to its sigmoidally normalized equivalent as in (4.8):

$2\log O(p,q)$	$Sig(\log O(p,q))$	Evidence for p over q
0 to 2	.5 to .73	Not worth more than a bare mention
2 to 5	.73 to .92	Positive
5 to 10	.92 to .99	Strong
> 10	$> .99$	Decisive

Thus, the sigmoidal function, which is commonly used in neural networks, can be seen as computing the Bayesian conditional probability of $P(p|x)$ based on the log odds of two mutually exclusive and exhaustive hypotheses. The resulting probability values can be interpreted in terms of the reasonable belief in the truth of one hypothesis over the other, as per the above table.

Individual Probability

Using the above equations, we can express the individual probability term as follows:

$$P(h_j^i|\mathbf{x}_p) = \frac{1}{1+e^{-\gamma^i \log \frac{P(\mathbf{x}_p|h_j^i)P(h_j^i)}{P(\mathbf{x}_p|\overline{h_j^i})P(\overline{h_j^i})}}} \tag{4.8}$$

Thus, this individual probability could be computed by a sigmoidal logistic function, with the net input (η_j) equivalent to:

$$\eta_j = \log \frac{P(\mathbf{x}_p|h_j^i)P(h_j^i)}{P(\mathbf{x}_p|\overline{h_j^i})P(\overline{h_j^i})} \tag{4.9}$$

Hinton and Sejnowski (1983), using the same Bayesian framework, argued that, based on (4.9), one should consider the weight from a given input unit x_i to represent the log *likelihood* ratio for that input unit:

$$w_{ij} = \log \frac{P(x_i|h_j^i)}{P(x_i|\overline{h_j^i})} \tag{4.10}$$

with the bias weight θ_j taken to represent the log *prior odds* ratio:

$$\theta_j = \log \frac{P(h_j^i)}{P(\overline{h_j^i})} \tag{4.11}$$

If this were the case, then the standard sigmoidal net input function:

$$\eta_j = \sum_i x_i w_{ij} + \theta_j \tag{4.12}$$

would result in the appropriate Bayesian expression for the individual probability of the hypothesis given the data, assuming that the input units (x_i) are binary, and independent, so that a product over the individual cases can be computed:

$$\begin{aligned} \eta_j &= \sum_i x_i \log \frac{P(x_i|h_j^i)}{P(x_i|\overline{h_j^i})} + \log \frac{P(h_j^i)}{P(\overline{h_j^i})} \\ &= \log \left(\frac{P(h_j^i)}{P(\overline{h_j^i})} \prod_i \frac{P(x_i|h_j^i)}{P(x_i|\overline{h_j^i})} \right) \\ &= \log \frac{P(\mathbf{x}_p|h_j^i)P(h_j^i)}{P(\mathbf{x}_p|\overline{h_j^i})P(\overline{h_j^i})} \end{aligned} \tag{4.13}$$

While this formulation provides an important starting point, there are a couple of problems with this interpretation that need to be resolved in order to work well within the ReBel framework. These are addressed in a subsequent section of this chapter, along with other possible ways of defining this function, such as with the use of a Gaussian or related functions. However, for the time being, we can assume that the individual probability is computed according to the sigmoidal logistic function of the weights times the activations:

$$P(h_j^i|\mathbf{x}_p) = \frac{1}{1 + e^{-\gamma^i \eta_j}} \tag{4.14}$$

where η_j is computed as in (4.12).

Relative Probability

Using the Bayesian framework presented above, we can express the relative probability term as follows:

$$P(h_j^r|\mathbf{x}_p) = \frac{1}{1 + \left(\frac{P(\mathbf{x}_p|h_j^r)P(h_j^r)}{P(\mathbf{x}_p|h_q^r)P(h_q^r)} \right)^{-\gamma^r}} \tag{4.15}$$

Thus, in order to compute this value, we need to define the component terms. There are two central questions — how to evaluate the likelihood term $P(\mathbf{x}_p|h_j^r)$, and how to define the null hypothesis h_j^r such that the terms involving it can be computed.

The definition of the likelihood term is something that gives the probability of the input vector $\mathbf{x}_p$ under the hypothesis h_j^r. In other words, assuming that h_j^r is true, how likely is it that we would observe the input vector $\mathbf{x}_p$. Essentially, this likelihood term ignores any of the relative comparisons between other hypotheses, and asks about the specific match between the input vector and the hypothesis. Thus, it makes sense to adopt the following definition of the likelihood of the relative evaluation of the hypothesis in terms of the individual probability of that hypothesis:

$$P(\mathbf{x}_p|h_j^r) \equiv P(h_j^i|\mathbf{x}_p) \tag{4.16}$$

given that the individual probability provides the relevant information, and it is easily computable as described above. In probabilistic terms, this means that the likelihood used for the relative probability term is the same as that used for the individual term, with the inclusion of he prior probability of the individual probability term (and the probability of the data itself, but this will cancel out in the odds ratio anyway):

$$P(h_j^i|\mathbf{x}_p) \propto P(\mathbf{x}_p|h_j^i)P(h_j^i) \tag{4.17}$$

Recall that this individual prior is effectively implemented by the bias weight of the unit. This inclusion of the prior probability of the individual hypothesis in the relative likelihood is appropriate assuming that we want our relative comparison to take into account the individual priors over the different hypotheses we are considering, which seems much more reasonable than *not* including them, for example.

Next, we need to define the null hypothesis h_q^r. The interpretation of h_q^r as an "event" that can be described probabilistically can be given as the probability that a middling hypothesis among the set H of other hypotheses in the layer accounts for the data. With the kWTA assumption used here, a middling hypothesis can be defined as one that is less probable than the k most probable hypotheses, but more probable than the next most probable hypothesis:

$$P(h_q^r) \equiv P(h_{k+1}^r) + q[P(h_k^r) - P(h_{k+1}^r)] \tag{4.18}$$

where $(0 < q < 1)$ and the subscripts denote a rank-ordering of the hypotheses in the layer from 1 (most probable) to N (least probable). The definition in (4.18) can be thought of as a functional form which can be used to define various conditional probabilities of h_q^r, for example:

$$P(h_q^r|\mathbf{x}_p) \equiv P(h_{k+1}^r|\mathbf{x}_p) + q[P(h_k^r|\mathbf{x}_p) - P(h_{k+1}^r|\mathbf{x}_p)] \tag{4.19}$$

The $P(h_q^r|\mathbf{x}_p)$ notation will be used in what follows for simplicity.

It is important to note that it is possible to define h_q^r in ways that do not depend on specifying an exact k parameter. In particular, I have used an alternative definition of h_q^r as a function of the mean

$(\overline{P(h^r)})$ and max $(\max P(h^r))$ probability values over the units in the layer:

$$P(h^r_q) \equiv \overline{P(h^r)} + q[\max P(h^r) - \overline{P(h^r)}] \tag{4.20}$$

with a q value of -0.1 providing a roughly 25% activity level. While this way of defining h^r_q avoids the need to specify a precise activity level, it ends up performing slightly worse than the "standard" kWTA definition (4.18) on the tasks reported later in this thesis. Nevertheless, it is possible that this or some other way of defining h^r_q might be preferable in other tasks.

By using the definition of the likelihood in (4.16) and functional form of (4.18) for the null hypothesis terms, we have addressed most of the terms need to compute the relative probability. The single remaining issue is that of the prior probabilities, $P(h^r_j)$. These priors can be initialized to uniform values at the start of processing for a given input pattern, and then iteratively updated as a function of the previous ReBel belief state. This provides a novel form of settling, which works quite effectively, resulting in typically between 20 to 40 cycles of iterative updating to achieve a stable probability state (to a threshold of .02 difference between updates) for the entire network.

To summarize, the ReBel function can be computed by evaluating the individual probability using a sigmoidal logistic function of the net input to the unit, and computing the null hypothesis as a function of these individual probabilities over the layer. The prior probabilities are updated over settling as just described. Thus, the ReBel function is as follows:

$$ReBel(h_j, \mathbf{x}_p) = P(h^i_j|\mathbf{x}_p)\frac{1}{1 + \left(\frac{P(h^i_j|\mathbf{x}_p)P(h^r_j)}{P(h^i_q|\mathbf{x}_p)P(h^r_q)}\right)^{-\gamma^r}} \tag{4.21}$$

The appendix *Implementational Details of LEABRA* describes a number of additional details regarding the implementation of the ReBel activation function that make it work well in practice. These details include mechanisms that keep the sigmoidal functions in a useful dynamic range, preserve small activation values for learning, and compensate for overall decreases in activation as uncertainty propagates through a deep network. In addition, the way in which external inputs are clamped and the use of thresholds and other techniques to improve computational speed are discussed.

Functional Properties of ReBel

There are several important functional properties of the soft kWTA activity constraint imposed by the ReBel function. As discussed above, this constraint will cause individual units to develop more specialized representations, which is consistent with the entropy reduction principle. This will be a primary issue of investigation in the simulations reported later. In addition, there are couple of less obvious implications of ReBel. One, which was mentioned briefly above, is that ReBel assumes that the unit activity states represent an underlying *binary* variable, which corresponds to whether or not a particular hypothesis applies to the current input pattern. Thus, it is not appropriate to use

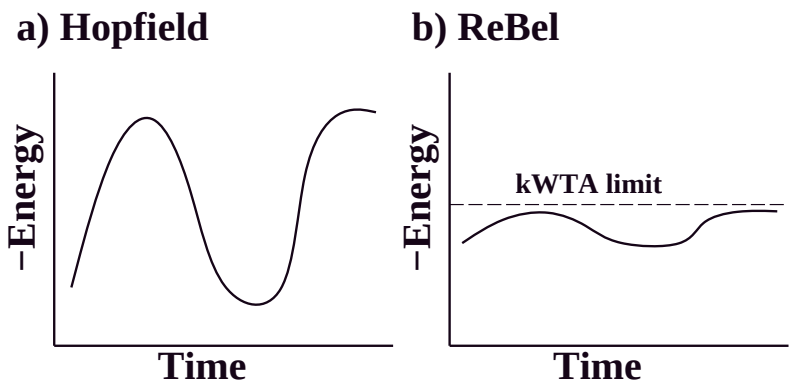

Figure 4.4: Damping effect of the ReBel activity constraints. **a)** Shows a hypothetical graph of the negative of the energy (a.k.a. "goodness") of the activation state of a continuous Hopfield network over time (settling). **b)** Shows a similar hypothetical graph for the ReBel activation function, which effectively has an upper limit on the total energy due to the kWTA limit. Thus, the net result of ReBel is to damp the activation dynamics compared to the Hopfield network.

this function in tasks that require the representation of continuous-valued quantities in single units (of course, it is possible to spread continuous values over multiple bins, each represented by an individual binary unit). It is easier to justify the use of binary representations on biological grounds than continuous-valued ones. First of all, neural firing is quite noisy, and thus unlikely to be precise enough to be encoding continuous values. Secondly, there is evidence that in situations where continuous values are needed, such as the representation of angular direction in motor control, the brain appears to use a population code (Georgopoulos, 1990), where the continuous value is represented as a weighted-average over a large number of individual units, each of which represents a particular direction. Thus, the neocortex appears to use, at least in this case, a probabilistic representation of underlying binary variables.

Another interesting property of ReBel is that the kWTA activation constraint provides a kind of *damping effect* on the activation dynamics of the network. This is illustrated in cartoon form in Figure 4.4, which shows two hypothetical plots of the energy functions for a continuous Hopfield network (Hopfield, 1984) and a ReBel network. The Hopfield network provides the activation function for the deterministic Boltzmann machine, GeneRec, and the Almeida-Pineda algorithm. The energy function is a quadratic scalar measure of the activation state, which is related to the interpretation of a network as a physical system, and can be used to derive learning rules (Ackley et al., 1985) and to show that the network will settle into a stable equilibrium state (Hopfield, 1982, 1984). The energy function is:

$$E = -\frac{1}{2}\sum_j \sum_i a_i w_{ij} a_j + \sum_j H(a_j) \tag{4.22}$$

where $H(a_j)$ is a measure of the entropy of the activity state. It is often easier to think in terms of the negative of this function, which is also known as "goodness," since this value increases with increasing activity levels of the units, as is represented in the figure. Thus, the goodness of the activation

state in ReBel has an upper bound imposed by the activation constraints (for a given set of weights). This upper bound, and the general tendency of ReBel to maintain activity around the kWTA limit, serves to restrict the range of energy states that the network can explore over settling. This restriction or damping of the activation dynamics might make the network less sensitive to changes in input patterns. As discussed previously, an important problem with interactive networks (like the Hopfield network) is that they are overly sensitive to small differences in input patterns, and that this contributes to their poor generalization performance. Thus, the damping effect of ReBel could be important in this respect, as will be explored in the simulations reported later. Chapter 8 discusses some other related consequences of the activity constraints imposed by ReBel.

Biological Properties of ReBel

The ReBel activation function can be interpreted as a summarization of the effects of the elaborate inhibitory interneuron system of the neocortex on the activity of pyramidal cells. The null hypothesis term which is a function of h_q^r serves as the summary inhibitory signal for the units in ReBel. The way in which this inhibitory signal is used in ReBel is consistent with several facts about inhibition in real neurons. First, inhibitory current in neurons has a divisive or *shunting* effect, since the membrane potential V_m (in a passive system, ignoring spatial gradients) is given by:

$$\frac{1}{\lambda}\frac{dV_m}{dt} = \left[\frac{g_e}{g_e + g_i + g_l}(V_e - V_m) + \frac{g_i}{g_e + g_i + g_l}(V_i - V_m) + \frac{g_l}{g_e + g_i + g_l}(V_l - V_m)\right] - V_m \tag{4.23}$$

where g_e is the excitatory conductance, g_i is the inhibitory conductance, and g_l is the leak current. The main effect of increased inhibition is to cause the denominator (which is the sum of all conductances) to grow, effectively dividing the impact of the excitatory conductance in the first term of (4.23). Similarly, the inhibitory term which is a function of h_q^r in ReBel divides the excitatory term.

Another property of ReBel that is consistent with the biology is that the value of the probability of h_q^r floats with the total level of excitatory activity in the layer, and is common to all the units in the layer. Similarly, the inhibitory interneurons project diffusely to many pyramidal cells in the local area, and they are in turn activated by pyramidal cells within the same local area, and by excitatory inputs coming into the area from other regions. However, it is important to note that ReBel does not have much to say about the details of how this feedforward and feedback inhibitory circuitry results in a floating inhibitory threshold like that based on h_q^r. Indeed, this mechanism might be somewhat complicated (Gibson et al., 1991), and substantial detailed modeling might be necessary to understand how it works. Given this, ReBel can be seen as an approximation of the net effect of the more detailed biological inhibitory mechanisms.

Individual Probability Functions for ReBel

The computation of the ReBel activation function depends entirely on the evaluation of the individual probability term, $P(h_j^i|\mathbf{x}_p)$, which is computed as a function of a unit's weights and the

activities of the sending units. These individual probability values are similar to the activation values computed in a standard neural network (e.g., backpropagation), and it was shown above that the sigmoidal logistic function computes this individual probability under certain interpretations of the weights and sending unit activations. However, there is another commonly used neural network activation function that could potentially serve as a definition of the individual probability term for ReBel: the Gaussian. The sigmoid and Gaussian functions bring with them two distinct sets of underlying assumptions for the meaning of the weights and their relationship with the sending units. This section will explore which of these assumptions (or which aspects of them) are more or less appropriate for use in LEABRA.

In order to understand the applicability of these functions in the LEABRA framework, we must first be clear about the constraints and interpretations imposed on the unit activities and weights by LEABRA. In order to be consistent and allow for multi-layered networks, the sending units are assumed to be like the hidden units described above. Thus, they have an activity state which reflects a belief measure for the truth of some hypothesis represented by that unit, ranging between 0 and 1. The exact interpretation of the weights depends on the individual probability function, but, consistent with the LEABRA framework, we will consider them to also be bounded in the 0 to 1 range, and use them as the single parameter that determines the influence of the truth value of the sending unit's hypothesis on the receiving unit's hypothesis. In order to fully specify the assumptions of the individual probability function, we need to know: 1) the nature of the relationship between the truth value of a given sending hypothesis and the truth value of the receiving hypothesis; 2) how the weight value modifies, interacts, or otherwise specifies this relationship; 3) how the individual sending hypothesis contributions are aggregated to obtain an overall probability for the receiving unit.

These three points will be explored for the sigmoid and Gaussian functions below. It will be shown that these two functions can be seen as specialized for implementing the two main representational principles behind LEABRA, entropy reduction and information preservation, respectively. A new individual probability function that provides a reasonable tradeoff between both entropy reduction and information preservation, called *GausSig*, is then described.

It is important to be clear about the formal status of these functions, since they are described using a probabilistic framework, but are only really required to provide a continuous-valued number between 0 and 1 that reflects the truth value or degree of belief in the hypothesis given the input data. Thus, where there are probability distributions involved in the derivation (as in the Gaussian case), it is assumed that expected values are used. Further, it is important not to become overly restricted by the constraints of the probabilistic framework. Thus, while the sigmoid and the Bernoulli (which can be viewed as a more correct variant of the Gaussian for binary variables) are formally correct as probability functions, the Gaussian and GausSig are not. Nevertheless, these functions each provide a novel, useful, and well-defined framework for understanding the relationship between the truth value of sending units and a receiving unit as mediated by its weights. The lack of formal rigor

for these functions is also mitigated by the fact that they are closely related to functions which are formally correct. Thus, their formal properties can be understood in reference to these functions. In terms of actual network performance, the GausSig function clearly works the best. This section conveys the reasons why this should be so, but falls short of proving the case mathematically.

The Sigmoidal Logistic Function

As described previously, the sigmoidal function computes $P(h^i_j|\mathbf{x}_p)$ if each weight is equal to the likelihood ratio of the sending unit under the hypothesis (compared to the the hypothesis being false), and the sending units are binary and independent. The previous definition of the bias weight as the prior ratio in (4.11) is not problematic, and will not be discussed further. Another way of viewing the likelihood ratio which relaxes some of these assumptions is to write the equation the other way around as a definition:

$$\log \frac{P(x_i|h^i_j)}{P(x_i|\overline{h^i_j})} \equiv x_i w_{ij} \tag{4.24}$$

where we are now defining the evaluation of the log likelihood ratio to be the product of the sending unit belief value (a.k.a. activation) times the weights. This allows the sending units to be continuous-valued, but they still must be independent, which is probably an assumption that must be lived with. In any case, this definition of the likelihood ratio has important implications for interpretation of the impact of the sending unit's truth value on the receiver's, and the way in which the weight mediates this relationship.

Keeping in mind the sign constraints imposed by LEABRA, (4.24) implies that the contribution of a sending unit to the likelihood of the receiver is either positive or zero[1]. Further, the weight can only determine the degree to which the truth value of the sender supports the truth value of the receiver. This relationship can be understood in terms of the *graded implication* of the sending hypothesis on the receiving one. Thus, the individual relationship between the truth value of the sending hypothesis on the receiving hypothesis is one of logical implication, which can be denoted $x_i \rightarrow h^i_j$ for sending hypothesis x_i and receiving hypothesis h^i_j. The implication relationship is also known as a *conditional* relationship. That is, *if* the sending hypothesis is true, this implies that the receiving hypothesis is true as well (which is the familiar *modus ponens* form of logical deduction). Logical implication also means that if the sending hypothesis is false, this implies *nothing* about the truth of the receiving hypothesis (which people tend to find counterintuitive)[2]. Unlike the discrete logical case, the sigmoid function is *graded*, with the implication relationship mediated by the strength of the weight. This can be written $x_i \stackrel{w_{ij}}{\rightarrow} h^i_j$, which means that the truth value of the sending unit con-

[1] Note that prior probabilities (i.e., bias weights) should be rather low, so that some amount of sending input support is necessary to result in a high truth value on the receiver.

[2] Note also that the *modus tollens* form of reasoning, which allows one to conclude that x_i is false if h^i_j is false, does not necessarily apply since there are many other factors that contribute to the truth value of h^i_j besides x_i, and these factors could cause h^i_j to be false while x_i is true (and the weight between them is strong). It is interesting to note that people have difficulty applying *modus tollens*.

tributes to the truth value of the receiving unit as a graded function of the magnitude of the weight.

A particularly important aspect of this graded implication relationship is that it allows for a sending hypothesis to have graded levels of relevance in proportion to the weight, with the extreme of complete irrelevance occurring when $w_{ij} = 0$. This is important because it allows a receiving unit to perform entropy reduction, which is of central importance for the LEABRA framework. There are two primary forms of entropy reduction that the sigmoid facilitates, filtering and categorization. Filtering can be performed by simply having zero weights from a set of sending units. This means that the truth value of the these sending units is irrelevant for the receiving unit. Generally speaking, zero-valued weights enable the receiving unit to develop representations that focus on separate features present in the input, and ignore other features. Categorization can be performed by having the same (non-zero) weight value for a set of different sending units, which means that each such unit will have the same truth-value implication for the receiving unit. Assuming a ReBel-like activity constraint over these units, and by virtue of the fact that sending units with 0 truth values have no impact on the receiver, any subset of active inputs will result in roughly the same receiving truth value. This can be thought of as computing a logical OR over these sending units, which is the essence of categorization. It is easy to see that both filtering and categorization reduce the entropy of the receiving unit relative to that of the sending input pattern.

It is also important to note that the form of this function is biologically plausible, in that the sending unit contributes to the activation of the receiver in proportion to its activity, which is a reasonable approximation of what happens in biological neurons. Similarly, the weight plays a plausible role as synaptic efficacy, where greater impact of the presynaptic neuron is felt from larger weights. Finally, when the weight goes to zero, the connection could be "pruned" without altering the relationship between senders and receivers, which provides a potential basis for the important biological phenomenon of synaptic loss.

The Gaussian and Related Individual Probability Functions

While the sigmoidal function can be understood in terms of graded logical implication, it is most natural to think of the Gaussian function in *probability matching* terms. Thus, the weights of a given receiving unit specify that unit's expectation of what probability the sending unit will have whenever the receiver is active. The truth value of the receiving unit is high whenever the sending units are close to this expected probability, and lower when there is a discrepancy. This probability matching is also related to the idea that the truth value of the receiving unit should be a function of the probability with which it would have generated the pattern of sending unit activities (Nowlan, 1990). The probability matching idea is expressed in the Gaussian by virtue of it being based on the negative exponential of the squared distance between the weight and the sending unit probability:

$$P_{gauss}(h_j^i|x_i) = e^{-(x_i - w_{ij})^2} \tag{4.25}$$

Thus, unlike the sigmoid function, there is no pre-specified implication relationship between the truth of the sending unit's hypothesis and the receiver's. For example, if the sending unit is false (near zero) and the weight is also near zero, the distance will be quite small, and a high truth value will result. In other words, the receiving hypothesis can obtain support from the sending hypothesis being false if it expects it to be false. This is not possible under the logical implication relationship implied by the sigmoid with the positive-only weights and activations used in the ReBel framework.

One might describe the Gaussian relationship as being *biconditional* or *if and only if* (*iff*), in contrast to the conditional relationship of the sigmoid, with the direction of relationship specified by the weight value. When the weight is near 1, for example, the relationship between the hypotheses is $x_i \leftrightarrow h_j^i$, so that if the sending hypothesis x_i is true, then it should influence h_j^i to be true, and if x_i is false, it should influence h_j^i to be false[3]. An opposite case holds with a weight near 0, which implies $\neg x_i \leftrightarrow h_j^i$. Intermediate weight values do not imply graded versions of these cases — instead they imply an expected probability equal to the weight value. Thus, in contrast with the sigmoid, the weights in a Gaussian function specify the nature of the relationship between x_i and h_j^i, but not the graded strength of that relationship. Furthermore, the Gaussian enables the sending hypothesis to contribute to the receiving hypothesis being true or false, whereas the sigmoid (under the sign constraints present in ReBel) only allows the sending hypothesis to contribute to the truth of the receiving hypothesis.

Thus, the Gaussian is loosely speaking more *informative* than the sigmoid with respect to the relationship between the sending and receiving hypotheses. However, it is not very biologically plausible, because it violates many of the basic properties of neural interaction that are captured by the sigmoid. Thus, sending neurons under the Gaussian should be able to excite a receiving neuron with a small weights when they are not active. Further, as this sending neuron becomes more active, it should activate the receiver less. Neither of these properties is likely to be true of pairwise interactions between excitatory neurons in the cortex. While one might be able to construct networks involving inhibitory neurons where something approximating a Gaussian function could be implemented, the individual probability function is intended to describe the simple pairwise interaction between excitatory neurons, while the relative component of the ReBel function describes the effects of the inhibitory neurons.

To complete the description of the Gaussian, the individual distance contributions of the sending units are combined in a multi- dimensional Gaussian function, which, like the sigmoid, is based on a product over the individual terms. In this case, the product is justified by assuming that the contributions of the sending units are independent of each other. This product turns into an addition as a

[3] Again, it is not the case that the truth value of the receiver h_j^i can be used to reason about the truth value of the sender x_i.

result of the exponential function, resulting in:

$$\begin{aligned} \eta_j &= \frac{\sum_i (x_i - w_{ij})^2}{2\sigma^2} \\ P_{gauss}(h_j^i|\mathbf{x}) &= \frac{1}{K} e^{-\eta_j} \end{aligned} \tag{4.26}$$

where σ is the standard deviation of the Gaussian, and K is a normalizing term that is not relevant for use in ReBel. Note that it is possible to make this σ an additional parameter per connection (like the weight), which would allow the Gaussian to also express the effective strength of the relationship — a large σ means the sender has a weak impact, and a small σ gives a strong impact. The use of this additional parameter is not particularly biologically plausible, however, and it requires its own learning rule.

The Gaussian computes the difference between the expected sending probabilities (encoded by the weights) and the observed probabilities in a completely uniform manner. This is appropriate for continuous-valued variables which are uniformly distributed along a continuum. A related function which is more appropriate for binary-valued variables such as the sending unit hypotheses assumes Bernoulli-distributed variables:

$$P_{bernoulli}(h_j^i|x_i) = w_{ij}^{x_i}(1 - w_{ij})^{(1-x_i)} \tag{4.27}$$

This can be compared to the distance measure used in the Gaussian function. As was the case with the Gaussian, the probability of the receiver under this function is higher when there is a match between the weights and the sending activations. However, unlike the Gaussian, it is not maximal for all matching values of weight and activation. When the weights indicate that the sending unit should be off or on with high certainty (e.g., values around .01 or .99 respectively), then matching values of the activation result in a high likelihood measure (near 1). As the weight and activation decrease in certainty towards .5, the probability also approaches .5 (see Figure 4.5).

Finally, as with the Gaussian, the individual Bernoulli probabilities are aggregated by assuming independence and computing the product over all of them:

$$P_{bernoulli}(h_j^i|\mathbf{x}) = \prod_i w_{ij}^{x_i}(1 - w_{ij})^{(1-x_i)} \tag{4.28}$$

The Entropy Reduction, Information Preservation Tradeoff and the GausSig Function

The differences between the sigmoid and the Gaussian can be described in terms of entropy reduction and information preservation. As described above, the sigmoid naturally allows the entropy-reducing filtering (near zero weights) and categorization (logical OR type of relationship) to be encoded. In contrast, the weights of the Gaussian (or Bernoulli) determine the specific probabilities of the sending units that the receiver expects, and the resulting probability provides maximal informa-

tion regarding the fit between this expectation and the actual state of the sending units. However, this information comes at the cost of being unable to perform entropy reduction via filtering or categorization. Thus, the sigmoid and Gaussian functions can be seen as optimal for the entropy reduction and information preservation aspects, respectively, of the overall representational objectives behind LEABRA. As was the case with the MaxIn learning rule, it may be possible to combine these two into one unified function, which provides a good tradeoff between the advantages and disadvantages of both. Such a function, called *GausSig*, is described below.

Before describing GausSig, it is useful to consider some of the more practical aspects of the sigmoid and Gaussian functions in the context of the ReBel framework. The most relevant aspect of the ReBel framework here is that individual units compete with each other on the basis of their individual probability values for the ability to represent input patterns. Thus, it is important that the individual probability function encourage a useful form of competition among the units. This is where the informativeness of the Gaussian function has a distinct advantage over the sigmoid function, since it reflects the goodness-of-fit between the input pattern and the weights. It is impossible for a Gaussian unit to fit a large number of different input patterns very well, since weights that are close to one set of input patterns will necessarily be further away from dissimilar inputs. For this reason, Gaussian units will do a good job of sharing the representational space across many units. In contrast, it is quite easy for a sigmoidal unit to match a large number of different patterns, since larger weights always result in a higher probability value, and a unit with large weights to many inputs will become consistently activated. On the other hand, it is difficult for a reasonable number of Gaussian units to span a high-dimensional input space (especially with random initial weight values), since each unit is unable to cover enough territory within the space, whereas sigmoidal units more naturally cover high-dimensional spaces, even with random weights.

Thus, the greater information preservation of the Gaussian function leads to good differentiation of the units in a competitive situation (it is hard for units to dominate the representational space), but it simultaneously causes units to become overly-specific, and unable to cover high dimensional spaces easily. In contrast, the entropy reduction properties of the sigmoid can result in poor differentiation of the units in a competitive situation, but they can more easily carve up a high dimensional space. Thus, at a practical level, it seems clear that something in between the sigmoid and the Gaussian would be desirable.

The idea behind the GausSig function is to retain the broad features of the sigmoid function, given that it is biologically plausible and that entropy reduction is of paramount importance in LEABRA, but to introduce a Gaussian-like distance term into the computation of the likelihood ratio. This distance term will introduce some degree of sensitivity to the match between the weight and the sending unit probability, enabling the unit's likelihood to be a more informative function of its weights. Thus,

the value of the likelihood ratio under GausSig is:

$$\log \frac{P(x_i|h_j^i)}{P(x_i|\overline{h_j^i})} \equiv x_i w_{ij}(1 - k_{gauss}(x_i - w_{ij})^2) \tag{4.29}$$

where k_{gauss} determines the overall scaling of the distance term (maximum reasonable value is 4, 2 is typically used). This term is the same as the standard sigmoid at the important boundary conditions when either $x_i = 0$ or $w_{ij} = 0$, which preserves the important entropy reduction characteristics of the sigmoid, and when $x_i = w_{ij}$. However, outside of these values, the likelihood is lower, due to the distance term. Thus, it is important that the distance factor in GausSig is multiplied by both x_i and w_{ij}, since this preserves important features of the sigmoid, and retains the idea that the weights reflect the importance of the sending unit on the receiver. What GausSig adds over the sigmoid is the idea that the level of importance attributed to the sending unit should also match the expected probability of that sending unit as well. This view of the weight as reflecting both relevance and expected activation value is quite consistent with the MaxIn learning rule, as will be described below, where the ZSH component causes the weights to become either near 1 or near 0 as a function of the coactivation of the sending unit and the receiver, while the SCL component causes the weights to match the expected value of the sending unit.

Figure 4.5 shows plots of all four of the individual probability functions discussed in this section, for all values of a single input and weight. It is clear that GausSig exhibits a combination of the sigmoid and Gaussian properties, and further that its top half resembles the top half of the Bernoulli function. It is also clear that both the sigmoid and GausSig part company with the Gaussian and the Bernoulli in that small to zero values of weight and sending activation result in lower likelihood values for the receiver, not larger values. This is important for biological plausibility, as discussed above.

In some ways, GausSig is uniquely suited to the ReBel activation function, since it does not work very well as a standard activation function (e.g., in a backpropagation network). This is because GausSig assumes that the weights are bounded in the same range as the activations (typically 0 to 1), since there is some pressure for them to match the activation values. As discussed above, this results in a strictly positive net input that would have to be compensated for by carefully chosen negative bias terms to result in an informative pattern of activation over a set of units. In simulations where the GausSig was substituted for the sigmoid in a standard backpropagation or GeneRec network, it did not work very well for this reason. However, the ReBel activation function in LEABRA automatically compensates for overall levels of activation and sets activation based on the relative activations of different units. Using the GausSig likelihood function in LEABRA was never found to impair learning on any of the tasks studied, and on most tasks it resulted in significantly faster and more reliable learning.

While the GausSig function might at first glance appear to represent a departure from biological plausibility, it can actually be justified based on the effect of synaptic activation on voltage and/or

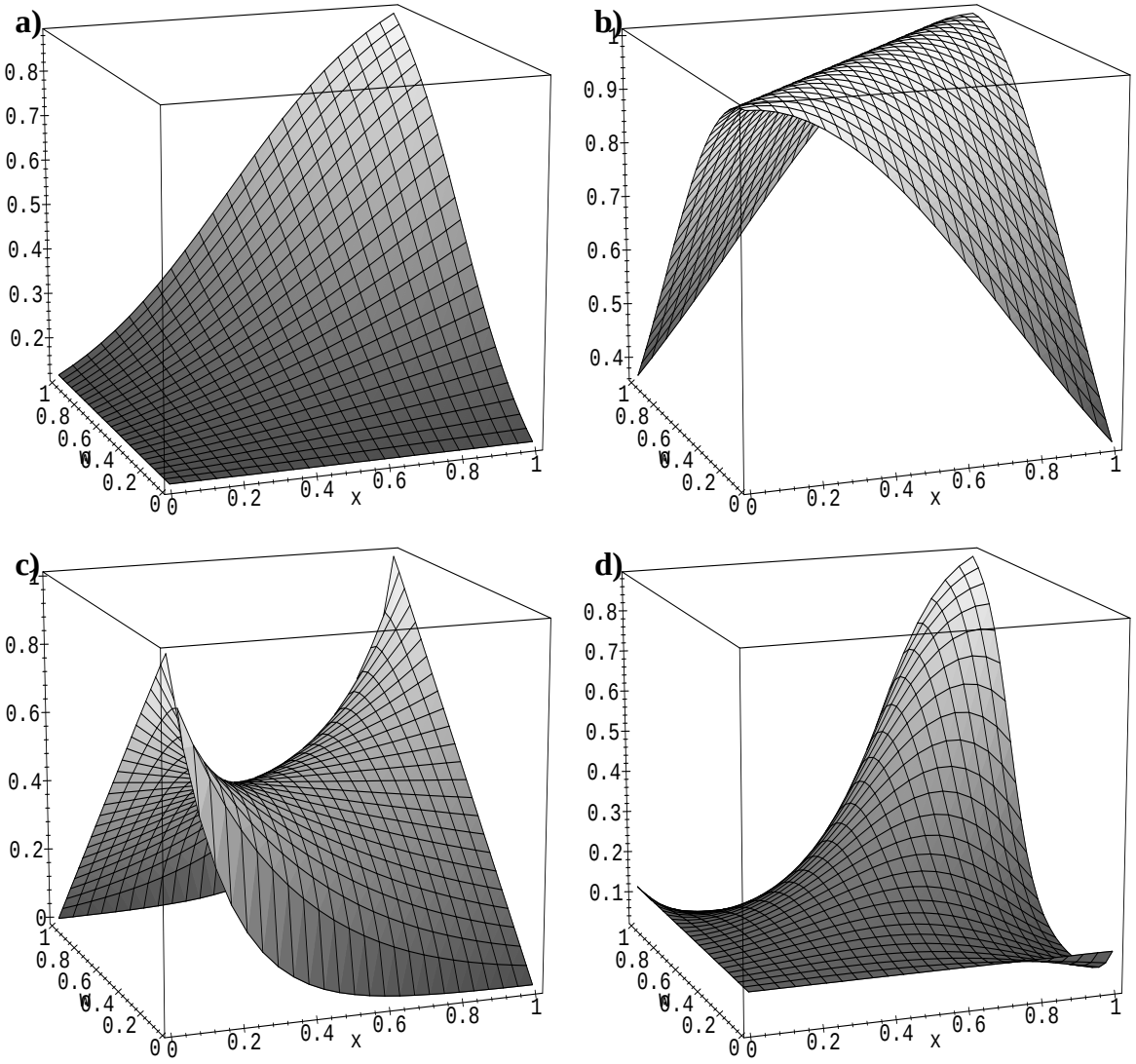

Figure 4.5: Four different individual probability functions plotted over the 0-1 range for a single sending activation (x) and weight (w). a) shows the sigmoid, with offset of .5 and gain of 4. b) shows the Gaussian, with standard deviation of 1. c) shows the Bernoulli. d) shows GausSig, with offset of .5, gain of 4, and k_{gauss} of 4.

calcium sensitive potassium channels. It is known that several forms of voltage and calcium sensitive potassium channels exist in cortical pyramidal neurons (Douglas & Martin, 1990), and that these channels provide a mechanism for adapting the firing thresholds of neurons in an activity-dependent manner. As the membrane potential or calcium level (which is determined in part by synaptic activity) of the neuron rises above some level, these channels are opened, and result in an increased leak current that brings the membrane potential back towards resting level. These channels could implement a GausSig-like function by generating increased leak current for weak synaptic inputs. Thus, when a synapse with a large weight (high synaptic conductance) is infrequently activated by the sending neuron, this results in a relatively large but transient increase in the local membrane potential, which could be enough to activate the potassium channels, resulting in a net inhibitory effect. Similarly, the frequent activation of a synapse with a small weight (weak conductance) would result in a sustained but weak depolarization, which might be enough to activate the potassium channels, but not enough to actually contribute sufficient excitation to overcome this increased leak current. Only in the case of a synapse with a large weight and a frequently firing sending unit would the increased potassium conductance be overcome by the synaptic excitation, resulting in a net depolarization of the cell. Thus, the match between sending unit activity (firing frequency) and synaptic weight (efficacy) emphasized in GausSig may also be emphasized in cortical neurons.

Finally, given that the net input under GausSig has a different kind of dependency on the weights and activations than a standard dot product, the derivative of this input term with respect to a weight is different. This derivative is used in the GeneRec error-driven learning rule, which should therefore be altered to use the new derivative whenever the GausSig function is used. The appendix on implementation details of LEABRA contains a derivation of the GeneRec learning rule for GausSig. Relatedly, the MaxIn learning rule is also influenced by the choice of individual probability function, as will be discussed below.

Entropy Reduction and Information Preservation: MaxIn Learning

Of the three proposed representational principles on the form of useful information processing, the standard representational form principle is most effectively handled by the fixed constraints discussed above. However, both entropy reduction and information preservation probably need to be implemented with adapting constraints, as it is difficult to satisfy these opposing forces in an optimal way without using an adaptive algorithm that takes advantage of the particular features of specific environments. The entropy reduction imposed by the ReBel activation function can be thought of as a first pass at this constraint, which can be further specified through adapting constraints.

There are potentially many different ways of implementing these constraints. A direct approach would involve formulating the information-theoretic measures corresponding to each constraint, and taking the derivative of these with respect to the weights in the network. These derivatives could then be used for a gradient descent learning procedure. However, a slightly more indirect approach

was taken because the relevant information-theoretic measures are not available locally in time (i.e., they are time-averaged values). The result of this approach is a self-organizing algorithm known as *MaxIn*, for *Maximizing Input Information*.

The motivations behind MaxIn and its derivation are given in O'Reilly (1994), which can be summarized as follows. Essentially, the problem with the direct approach suggested above is that the information-theoretic measures depend critically on the variance of the unit's output signal over time. However, as is argued in O'Reilly (1994), this introduces a temporal non-locality into the algorithm, the consequences of which are problematic for realistic learning environments where the statistics of the training sample from the environment are non-uniform with respect to the relevant categories and distinctions in that environment. This non-uniformity affects the validity of the output unit signal variance measured over intervals shorter than the grain size of the non-uniformity, and O'Reilly (1994) showed that this hurt algorithms that are derived on the basis of the output signal variance.

An example of this problem described in O'Reilly (1994) is the case of classifying different tree types. In the real world, the kinds of trees one sees across different climates are not very uniformly distributed. Each climate zone tends to have associated with it a different set of trees, from palm trees to pine trees. Thus, if one were to spend several months vacation in the tropics, where palm trees abound, the units representing these kinds of trees would be firing quite often, and those for the coniferous trees would be silent. Then, upon traveling to the Rocky Mountains for the remainder of the year, the reverse would occur. If one imagines that the entire space of trees is to be represented, then the appropriate time period to sample variables over is an entire year. However, this is clearly too long to wait to learn something new. With shorter time samples, the clustering of category instances over time will bias estimates of the mean and variance of unit firing rates, which might compromise the learning algorithm.

The alternative approach taken in MaxIn is to maximize the quality of the *input* signal information, as opposed to the output information. An example of this form of learning is the *G-maximization* algorithm (Pearlmutter & Hinton, 1986), which uses a sampling method across different phases of learning to measure and enhance the response of a unit to meaningful (i.e., correlated) input signals relative to their response to noise. Input information quality in MaxIn is defined in a similar way using the ReBel framework presented above. This is done by making the relative probability term a comparison between the hypothesis given a particular input signal $\mathbf{x}_p$ relative to the probability of that hypothesis when the input instead consists of a pure noise vector ν_p. Thus, the null hypothesis in this case, h_j^{ν}, is not a function of the other units in the layer as was the case with h_q^r in the activation function, but rather on the support given to hypothesis j by an input consisting of pure noise.

The probability of the null hypothesis is defined by the following functions, which simply sub-

stitute the pure noise input vector ν_p for the actual input vector $\mathbf{x}_p$:

$$\begin{aligned} P(h_j^\nu|\mathbf{x}_p) &\equiv P(h_j^h|\nu_p) \\ P(h_j^\nu) &\equiv P(h_j^h) \end{aligned} \tag{4.30}$$

By substituting h_j^ν for h_q^r in the ReBel equation (4.21), the MaxIn objective can be expressed formally as:

$$MaxIn(h_j, \mathbf{x}_p) = P(h_j^i|\mathbf{x}_p)\frac{1}{1+\left(\frac{P(h_j^i|\mathbf{x}_p)}{P(h_j^\nu|\mathbf{x}_p)}\right)^{-\gamma^r}} \tag{4.31}$$

Note that the prior probability of both the hypothesis and the null hypothesis is the same, so it cancels out in the odds ratio as is assumed by (4.31).

In order to actually compute h_j^ν directly without resorting to a computationally expensive sampling procedure, the noise vector ν_p can be defined as a function of the expected value statistics of the current input vector $\mathbf{x}_p$. First, it is assumed that the input signals are drawn from a binomial distribution with a parameter μ, which gives the expected probability that an input bit is active. μ can vary from pattern to pattern. The pure noise input vector ν_p is computed as if it came from the same distribution as that which generated the current input vector, so that $\mathbf{x}_p$ can be used as a sample from this distribution to estimate μ. In this case, the expected value of the activations across a given input pattern is an estimate of μ. Thus, the expected noise vector ν_p under these assumptions simply consists of a vector having each value as the expected value $\mu = \frac{1}{N_x}\sum_i x_i$:

$$\nu_p \equiv [\mu, ..., \mu] \tag{4.32}$$

Thus (4.32) and (4.30) can be used together with the definition of the likelihood function for the relative probability as in (4.16) to compute the $P(h_j^\nu|\mathbf{x}_p)$ term in (4.31). Also, it is assumed that $P(\nu_p) = P(\mathbf{x}_p)$ so that these terms cancel each other out in the odds ratio.

In order to obtain the MaxIn learning rule, the derivative of the MaxIn objective function (4.31) is taken with respect to the weights. This derivative can be used to perform gradient ascent in the input information signal conveyed by the weights. In order to compute the derivative, we express the MaxIn function in the following simplified terms:

$$p = P(h_j^i|\mathbf{x}_p) = f(\mathbf{x}, \mathbf{w_j}) \tag{4.33}$$

$$q = P(h_j^\nu|\mathbf{x}_p) = f(\mu, \mathbf{w_j}) \tag{4.34}$$

$$L = \frac{1}{1+\left(\frac{p}{q}\right)^{-\gamma^r}} \tag{4.35}$$

$$M = pL \tag{4.36}$$

resulting in a derivative of the following form:

$$\frac{dM}{dw_{ij}} = M\left[\gamma^r(1-L)\left(\frac{1}{p}\frac{dp}{dw_{ij}} - \frac{1}{q}\frac{dq}{dw_{ij}}\right) + \frac{1}{p}\frac{dp}{dw_{ij}}\right] \tag{4.37}$$

In order to compute the actual derivative, we need to decide upon the definition of the individual probability function $f()$. While this could be the same as the sigmoid or GausSig used to compute the activations, there is no reason to assume that it has to be the same, since this objective function is defining learning, not activation updating. Based on first principles, the Gaussian definition of the individual probability makes more sense for MaxIn, since it provides a more informative signal about the fit between the sending activations and the weights, which is exactly what MaxIn is attempting to optimize. Nevertheless, it is informative to compare the derivatives that result from using the sigmoid and the Gaussian.

When the derivatives of the p and q functions computed as sigmoids are substituted into (4.37), the result is:

$$\frac{dM}{dw_{ij}} = \gamma^i M[\gamma^r(1-L)((1-p)x_i - (1-q)\mu) + x_i(1-p)] \tag{4.38}$$

where γ^i is the gain of the individual sigmoid function, and γ^r is the gain of the relative one. Compare this with the result when a Gaussian definition of the individual probability is used, which results in:

$$\frac{dM}{dw_{ij}} = \frac{1}{\sigma^2}M[\gamma^r(1-L)(x_i - \mu) + (x_i - w_{ij})] \tag{4.39}$$

The two derivatives both have a term that is a function of the difference between the input and the mean $(x_i - \mu)$, with these terms modulated by the complement of the value of the corresponding p or q value in the sigmoid case. However, the sigmoid has an additional term which is always positive, which comes from the derivative of the sigmoid itself, which is always positive. Thus, this rule will result in the weights constantly getting larger, which increases the value of the p term (and thus the MaxIn objective function), but is not what is desired from an associative learning rule. In comparison, the Gaussian derivative is not always positive, and is a function of the difference between the input value and the weight. This has several desirable properties as discussed below. Thus, based on first principles and these derivatives, the Gaussian makes sense as a definition of the individual probability for the MaxIn learning rule.

While the MaxIn learning rule expressed in (4.39) has two key terms ($(x_i - \mu)$ and $(x_i - w_{ij})$) which will be discussed further below, it also has a couple of other terms which might appear somewhat more complicated (and potentially biologically implausible). The inverse variance constant $\frac{1}{\sigma^2}$ can be absorbed into the learning rate. The M term, which gives the value of the MaxIn objective function, could be computed directly. However, it could also be approximated based simply on the receiver's activation value. This simplifies the function considerably, and works well in practice. This leaves the $1 - L$ term, which is just the complement of the normalized odds ratio term in the

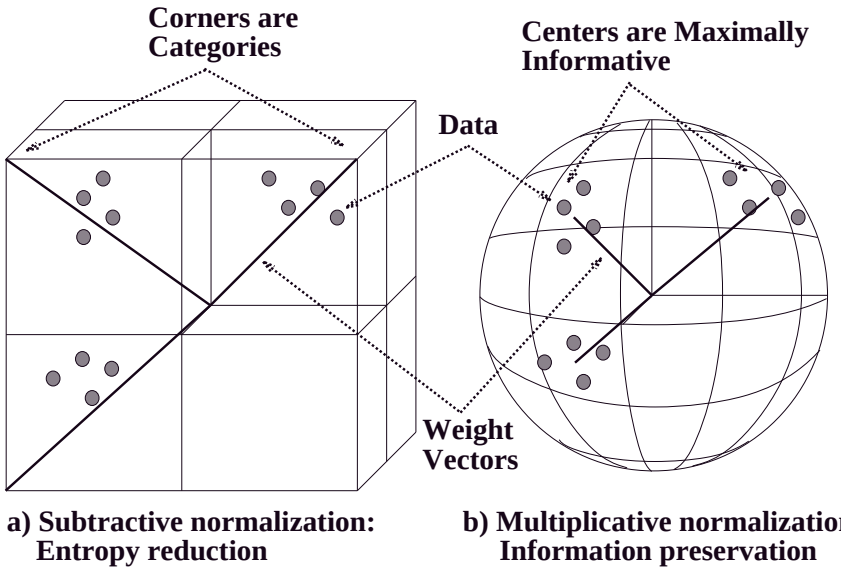

Figure 4.6: **a)** Subtractive weight normalization ($x_i - \mu$) yields weight vectors pointing at the corner of a (n-dimensional) hypercube closest to the input cluster. **b)** Multiplicative weight normalization ($x_i - w_{ij}$) yields (n-dimensional) spherical weight vectors pointed at the center of cluster of input patterns.

MaxIn learning rule:

$$L_c = 1 - \frac{1}{1 + \left(\frac{P(h_j^i|\mathbf{x}_p)}{P(h_j^\nu|\mathbf{x}_p)}\right)^{-\gamma^r}} \tag{4.40}$$

for which an easy approximation is not available, but which can be replaced with a constant without significant loss of performance (i.e., for biological plausibility). However, retaining it in the computational model does increase performance somewhat, for reasons that are discussed below. To summarize, the MaxIn learning rule can now be written in terms of a receiving activation value y_j, and optionally the above L_c term:

$$\frac{1}{\epsilon}\Delta w_{ij} = y_j\left[\gamma^r L_c(x_i - \mu) + (x_i - w_{ij})\right] \tag{4.41}$$

Interestingly, the two key terms referred to above each have important properties from the perspective of the entropy reduction and information preservation constraints. The entropy reduction constraint is evident in the first term ($x_i - \mu$) by virtue of its tendency to force the weights into a corner of the hypercube of possible weight states. For units which are often coactivated with the receiving unit, the term $x_i - \mu$ will usually be positive, and the weight will grow steadily towards its upper bound of 1. The opposite is true for units which are not typically coactivated with the receiver, and these weights will shrink steadily towards the lower bound of 0. Thus, the result is to force the weights into the maximum/minimum corners of the hypercube of possible weight states (see Figure 4.6a). This causes the units to form filtering, categorical representations, since differences among the inputs with near zero weights are filtered out, and many different patterns of inputs

on the near-one weights result in the same overall activation (categorization). As discussed earlier, filtering and categorization are both examples of entropy reduction, as they tend to throw away information about exemplars (inputs).

The information preservation constraint is implemented in the second term ($x_i - w_{ij}$) since it is directly related to the update rule that is obtained when applying the Linsker (1988) InfoMax objective, which is equivalent to the information preservation constraint. This term has the effect of moving the weight vector into the center of the input patterns which are coactivated with the receiving unit. This can be seen by iteratively computing the $x_i - w_{ij}$ term for all the different input patterns — the weight moves towards each input value, and, with a small learning rate, equilibriates around the expected value over all patterns (see Figure 4.6b). This expected value provides maximum information about the variance of any given input value, which is why this is information preserving.

Thus, the MaxIn learning rule, which is derived from the same assumptions as the ReBel activation function, provides an adapting implementation of the entropy reduction and information preservation principles. Another way of looking at the properties of MaxIn is in terms of the positive feedback problem associated with any kind of Hebbian associative learning algorithm. One way in which this problem is managed is via activation competition (e.g., ReBel). However, MaxIn has two versions of the other primary mechanism that counteracts the positive feedback problem, which is the normalization of the weight vector by subtractive and divisive constraints (Miller & MacKay, 1994; Goodhill & Barrow, 1994). The subtractive, entropy reduction term ($x_i - \mu$) has been derived in terms of a zero-sum Hebbian (ZSH) learning rule (O'Reilly & McClelland, 1992), where the sum of the weight values is maintained at a constant value by subtracting the mean weight change. The multiplicative, information preservation term ($x_i - w_{ij}$) is used in the soft competitive learning (SCL) rule (Nowlan, 1990). Thus, these components of MaxIn are also referred to as the ZSH and SCL components.

It is interesting that these two constraints correspond to the entropy reduction and information preservation constraints, respectively, and they also play a practical role in the ability of the network to form differentiated representations of the input space. Further, the MaxIn rule dynamically balances the tradeoff between these two learning components. This occurs because MaxIn weights the entropy reduction term in a manner proportional to the value of L_c in (4.40), which changes dynamically over learning as the ratio of the unit activity over the noise activity increases. Thus, as the unit develops a more discriminating (entropy reducing) response, the entropy reduction component of the learning is weighted less strongly, allowing the information preservation component to be emphasized more.

Auto-Encoder LEABRA: A Pressure to Encode Self-Structure

An important function of the MaxIn associative learning in LEABRA is to form useful representations of the correlational structure across patterns of activity to which it is exposed. This *self-*

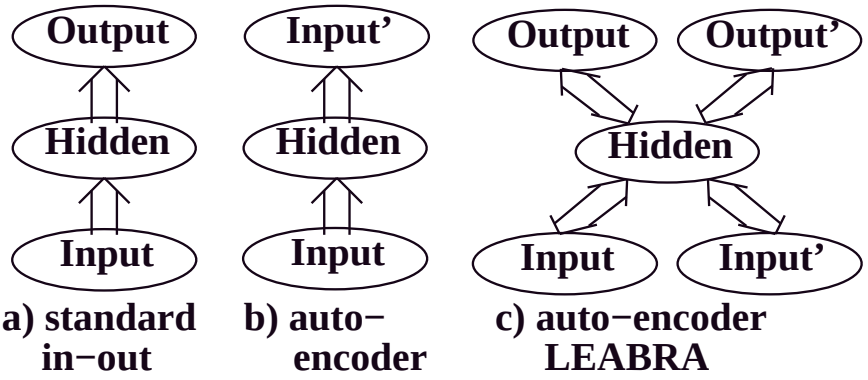

Figure 4.7: Illustration of the nature of the error-driven mappings performed in the auto-encoder form of LEABRA as compared with standard input-output and standard auto-encoder configurations. Primed notation indicates reconstruction. **a)** Shows the standard input-output mapping task, where the goal is to produce the correct output given a particular input. **b)** Shows the standard auto-encoder mapping, where the goal is to reproduce the current input pattern (the ′ notation indicates the reproduced version of that pattern). **c)** Shows the auto-encoder version of LEABRA, where the goal is to both produce the correct input-output mapping, and to reproduce both the current input and output patterns. Note that only one input and one output layer are used, but they have two states occurring sequentially in time as represented by the two layers in the figure.

structure across different activity patterns presented in the input and output layers of the network can be distinguished from the input-output mapping structure, which is what provides the usual error signals for error-driven learning. As described in the introduction, representing this self-structure contributes to LEABRA's good generalization performance, both within and between tasks. However, one of the central ideas behind LEABRA is that the combination of error-driven and associative learning is better than either alone. While the input-output mapping is shaped by both error-driven and associative learning in a standard LEABRA network, the representation of self-structure among the input and output patterns is not. Thus, this section describes a mechanism, called *auto-encoder* LEABRA (AE LEABRA), for introducing error-driven learning into the process of learning to represent self-structure. Note that the auto-encoder error signals are *in addition* to the standard input-output mapping error signals. This mechanism is similar to a standard auto-encoder, except that both the input and output patterns are reconstructed from the hidden activity pattern, by virtue of training the network to maintain these activity patterns in the absence of any input. It is also possible to apply a purely error-driven version of this mechanism in interactive, error-driven algorithms like GeneRec. The merits of this mechanism in both LEABRA and GeneRec are evaluated in the simulations reported in subsequent chapters.

Figure 4.7 shows the relationship between AE LEABRA and standard input-output and auto-encoder mapping tasks. Both of these standard tasks can be thought of as mapping an input to an output pattern, where the output in the case of the auto-encoder is simply the same pattern as the input. Thus, the network is trained in this case to reconstruct the input pattern via the hidden layer (which often represents a bottleneck, forcing a compressed representation there). In AE LEABRA, a standard input-output mapping is performed first (where the output is *not* a copy of the input), fol-

lowed by the task of reproducing the current input and output activity patterns given the current hidden unit state. Importantly, the reconstruction of the input and output patterns occurs *in the same layers* as those patterns originally occurred, and not in separate layers as in a traditional feedforward auto-encoder. This means that the error signals drive learning in the weights connecting the input and hidden layers, which are the same weights used in performing the input-output mapping task. Thus, the reconstruction task can also be thought of as maintaining the input and output patterns in the absence of any external input. The mismatch between the reconstructed (maintained) and the original input and output activity states serves as an error signal that, together with the error signal associated with producing the correct output given the input, and the Hebbian associative learning, drives learning. Because the reconstruction error signals are present in the same weights that drive processing of the input-output mapping task (due to the fact that patterns are reconstructed in the original layers), this learning should provide direct benefits on the mapping task.

The implementation of AE LEABRA is quite simple. First, the network performs the standard minus and plus phases of settling, where the minus phase has only the input clamped, and the plus phase has both the input and output clamped. The difference between these two states provides an error signal regarding the ability of the network to produce the correct output given the input pattern. Then, in the auto-encoder version, an additional *reconstruction* minus phase of settling is performed after the plus phase, with the activity values of the hidden units starting in their plus phase state, and no external input provided to any part of the network. This prior state provides the initial conditions for settling in the reconstruction phase, and reflects the hidden unit encoding of the original input-output activity pattern. If the hidden units have sufficiently encoded the input-output activity patterns, then they should be capable of reproducing them as the network settles during the reconstruction phase. In order to eliminate any signal from the actual input and output patterns themselves, the activity over these layers is zeroed prior to settling in the reconstruction phase. The reconstruction task can be made more difficult, and thus potentially provide a stronger learning signal, by decaying the prior activity state some fraction of the distance towards the mean activity level. Decay fractions from .5 to .8 appear to give the best performance on the various tasks.

It is also possible to implement the auto-encoder idea in standard error-driven networks, even non-interactive ones. In non- interactive, feedforward networks, one would need to have additional layers for the reconstruction targets, instead of performing the reconstruction in the original input and output layers. Thus, this method would not directly shape the same weights as used in the input-output mapping task, and would be expected to produce less significant results. In an interactive network like GeneRec, the same implementation as used in LEABRA can also be used. One simply retains the activity state from the previous plus-phase state as the initial conditions for setting in the reconstruction phase. Thus, one can compare the benefits of this additional auto-encoding pressure in purely error-driven networks with those for the LEABRA algorithm, where it operates in conjunction with Hebbian associative learning. Such comparisons are reported in the simulations later in this thesis.

Obviously, the introduction of this additional auto-encoder phase of training requires more of the biologically feasible implementation of this algorithm. In particular, it would require that the afferent input to sensory cortical areas (i.e., from the thalamus) has an alternating character, so that the cortical neurons are not driven very strongly by this input after having been driven. While something like this could feasibly be implemented in the biological system, I am not aware of data that either supports or refutes such an idea. Nevertheless, M. Hasselmo (personal communication) has suggested that cholinergic modulation of pyramidal neurons may specifically affect their sensitivity to extrinsic (non-cortical) inputs, which would be consistent with the needs of the auto-encoder idea. The biological implications of this auto-encoder idea are discussed in greater length in Chapter 8.

Finally, AE LEABRA can be compared to the *Helmholtz machine* proposed by Dayan et al. (1995), which is a self-supervised learning algorithm for developing hierarchical representations of environments, much like a standard auto-encoder. Unlike standard auto-encoders, and like AE LEABRA, the Helmholtz machine performs both recognition (stimulus-driven processing) and generation (reconstruction) over the same sets of units. However, while the same units are used in the Helmholtz machine, two different sets of weights are used. Thus, the central idea behind the Helmholtz machine is that the weights in the network can be divided into two parts — a set of bottom-up *recognition* weights, and set of top-down *generative* weights. In the "wake-sleep" version of this algorithm (Hinton, Dayan, Frey, & Neal, 1995), each set of weights is used to train the other. During the "waking" period, the recognition weights are used during processing of stimuli, and the resulting activation states used to train the generative weights so as to generate patterns of activity over the input layer like those observed during recognition. During the "sleeping" period, the opposite occurs: random activity patterns in the highest hidden layer are used to produce top-down activity patterns, which train the recognition weights to recognize stimuli consistent with this "generative model".

While both the Helmholtz machine and AE LEABRA share the goal of reconstructing the input using the same set of units, there are important differences between the two algorithms. 1) AE LEABRA performs both auto-encoding and input-output mapping tasks, while the Helmholtz machine only does auto-encoding. Thus, AE LEABRA is actually able to perform arbitrary tasks, while the Helmholtz machine is restricted to generating representations which may or may not be useful for solving a given task. 2) Both feedforward and feedback weights in AE LEABRA are active simultaneously for both activation propagation and learning, while the Helmholtz machine uses only one set of weights for activation propagation and the other for learning at a given point in time. This makes the Helmholtz machine inconsistent with all of the data regarding the role of interactive, bidirectional processing discussed in the introduction, and requires quite elaborate biological mechanisms to implement the use of different weights in different ways in the two phases. 3) AE LEABRA has all of the built-in self-organizing biases described above, while the Helmholtz machine is relatively unconstrained. Thus, the Helmholtz machine is likely to suffer from the same types of problems as purely error-driven learning algorithms like backpropagation and GeneRec.

Chapter 5

Part II: Computational Evaluation of LEABRA

This and the following chapters provide an empirical evaluation of the theoretical claims regarding the LEABRA model of learning in the neocortex developed in the preceding chapters. This evaluation is focused on the themes described in the introduction, which are summarized briefly below. The tasks used in the evaluation are selected in the context of a taxonomy of task variables relevant to these themes, together with a consideration of how the properties of LEABRA should interact with these variables. Thus, this taxonomy, which is described in detail below, provides a basic framework for understanding the simulations and the performance of different algorithms on them.

Following the description of the task taxonomy, a summary of the different algorithms and algorithm variants that will be used in the evaluation is presented. This includes the development of various alternative formulations of the basic ideas behind LEABRA which test some of the implementational principles and choices made in the standard LEABRA algorithm. In particular, the fixed (non adapting) activity constraint enforced by the ReBel activation function is compared against a set of adapting activity constraints derived below. Further, an attempt was made to construct an associative learning algorithm that worked with the adapting activity constraint, but it was not possible to get a system that worked reliably, for reasons that are explained below.

Finally, this chapter goes on to provide a general introduction to the basic computational issues in the context of a handwritten digit recognition task, which clearly illustrates the relative computational properties of LEABRA and the set of comparison algorithms. The next chapter explores the issue of generalization in greater detail in a simple combinatorial environment, which allows more precise parameterization of the generalization problem than the digit recognition task. The final chapter explores the issue of learning relational problems and learning in deep networks, using the family trees task and variants thereof.

Computational Themes Explored

Under-constrained representations in purely error-driven networks result in poor generalization in interactive networks: Because most problems insufficiently constrain the weights of a purely error-driven network, the representations developed by such a network tend to reflect the residue of the random initial weights as much as the constraints of the task. The ability of under-constrained weights to solve even complex tasks can be explained in terms of the diffusion of representation over many different hidden units. Each hidden unit makes a small contribution to representing the important regions of the input space, as described in the introduction. A feedforward network is often able to generalize quite well based on this kind of solution, as it exhibits a graded, proportional response based on the similarity of the novel input to known inputs. In contrast, an interactive network like the biologically feasible GeneRec error-driven algorithm described in Chapter 2 tends to be highly sensitive to small differences between input patterns (the butterfly effect), and thus does not generalize well. This sensitivity is a function of the turbulence (noise) in the activation space over which settling occurs, which tends to be high in under-constrained, error-driven networks. Thus, interactive error-driven networks will generalize significantly worse than similar feedforward networks on most tasks, and this should be correlated with the extent to which the network is under-constrained by the task. Further, the prediction that interactive networks are highly sensitive to small differences in input patterns can be directly tested.

Self-organizing constraints in LEABRA improve generalization: Since LEABRA uses the GeneRec learning algorithm, it is necessarily an interactive network, and should therefore be susceptible to the problems just described. However, the use of the Hebbian associative learning and activity competition in LEABRA, which together implement the self-organizing learning of useful representations according to the principles described in Chapter 3, should improve generalization. The activity competition causes individual units to take greater responsibility for representing particular input patterns, and the associative learning causes these representations to represent the correlational structure of the environment. Both of these should result in a smoother, less turbulent activation space that leads to better generalization. Further, the activity constraints themselves have a damping effect on the activation dynamics of the network, reducing its sensitivity. This should also contribute to better generalization.

LEABRA performs cross-task generalization better than error-driven algorithms: Purely error-driven networks tend to represent only the distinctions necessary to solve the particular task they are trained on, since it is only task-driven error that drives learning in these networks. Thus, they do not necessarily generalize well to novel tasks using the same environment. In contrast, the representations in LEABRA are constrained to represent the correlational structure of the environment, in addition to solving the particular task. Such representations should subserve better generalization to novel tasks in a given environment relative to purely error-driven networks.

LEABRA learns better than error-driven algorithms in deep networks: Purely error-driven algo-

rithms in networks with many hidden layers (deep networks) inevitably learn more slowly than those with just one hidden layer. This is largely because the error signals become more distant and indirect as more hidden layers are introduced, resulting in slower learning which often does not produce better final performance in terms of generalization or other measures. However, the self-organizing learning in LEABRA, which does not depend on potentially distant error signals, should result in faster learning in deep networks. Units at all layers in LEABRA should be developing useful representations according to the representational principles of entropy reduction and information preservation as implemented in the associative learning and activity constraints. The remote error signals thus play more of the role of *selecting* among potentially useful representations, rather than *creating* these useful representations in the first place.

Auto-encoder (AE) LEABRA performs better than regular LEABRA: By providing both error-driven and associative learning signals for the task of encoding the self-structure of the input and output patterns, the auto-encoder version of LEABRA should perform better in general than the standard version, where error signals are available only for the learning of the input-output mapping.

Fixed (non-adapting) constraints work better than adapting ones: Since adapting constraints can be overridden by other pressures during learning, and in general increase the number of parameters being adapted, they should provide less of a benefit compared to fixed constraints which truly constitute an *a priori* model.

LEABRA performs better than standard regularizers like weight decay and noise: While it is clearly impossible to compare LEABRA with the large number of alternative regularizing functions that have been developed, it is useful to compare it with a few of the standard ones. Thus, LEABRA will be compared with weight decay and the use of noise during training, which are two standard and reasonably biologically plausible ways of constraining error-driven learning. It is hypothesized that the associative learning and activity constraints in LEABRA, which are capable of generating useful representations even in the absence of error signals, will perform better than weight decay, which does not itself encourage the formation of useful representations (indeed, it leads to all-zero weights if unchecked by error signals). The use of noise is primarily useful for learning continuous-valued input-output mappings, where it serves to expand the range of continuous-valued space searched by the learning process. In contrast, LEABRA (and the cross-entropy formulation of backpropagation and GeneRec) are based on the idea that units represent binary values, and all of the problems tested are of this nature. It is hypothesized that noise will not provide much benefit in this case, as it is patterns of activity over many units, and not particular values of individual units, that are relevant in these problems. Thus, LEABRA should outperform both weight decay and noise.

A Taxonomy of Tasks

While it is the case that, at a very general level, all tasks amount to performing some form of input-to-output mapping, there are a number of different ways that this mapping can be structured.

The following is one taxonomy of distinctions among various tasks that are likely to be important for evaluating the performance of different algorithms. In particular, these distinctions are important for understanding on which tasks and in what way the associative learning and activity constraints built into LEABRA can facilitate learning. This provides a more detailed explication of the general themes described above, and constitutes an outline of the simulations.

There are two levels of distinctions: One level has to do with how a given item is represented in terms of a pattern of activity over units in the input or output layer (i.e., the *self-structure* of the items), and the other has to do with the overall task being performed (i.e., the *mapping structure* of the relationship between inputs and outputs). A given task can be described at both of these levels. For each different type of task feature, a discussion of the impact of the associative learning and activation constraints in LEABRA is presented. It is important to note that this taxonomy is not intended to be exhaustive, and the distinctions made are not necessarily mutually exclusive. Thus, a given task may have elements of several of the following properties. For example, the task of pronouncing written words has some relational elements (e.g., the final 'e' interacting with the vowel to make it long), while being substantially combinatorial, since many letters can be pronounced essentially independently.

Self-structure of Item Representations

Relational: The relationship between different units in a pattern is what identifies an item, not the activity of any specific unit or set of units. For example, in the handwritten digit recognition task, digits can appear in various locations in the input array, so that what is common to a given digit is the relationship between the pixels, not the specific pixels themselves. Another case of a relational input is when there are different input features that combine to mean different things. Thus, the meaning of a given item is defined by the relationship between the features present.

LEABRA can facilitate learning with relational stimuli in two ways. One is by virtue of the activity constraints, which can encourage different subsets of units to represent different combinations of a set of features. Thus, because units have a pressure to be more sparsely active, they will tend to pick up on conjunctions of input units or features, not on individual ones. This phenomenon has been studied extensively with respect to representations in the hippocampus, which has a very sparse level of activity (O'Reilly & McClelland, 1994). While the kWTA property of LEABRA can be made to reflect the extreme sparsity of the hippocampal system, it is likely that activity levels in the neocortex are considerably less sparse. Nevertheless, some degree of conjunctive encoding can be expected with the roughly 25% activity levels typically used in LEABRA simulations. The other way in which LEABRA can facilitate learning in these tasks is from well-known properties of Hebbian associative learning, which will pick up on the correlations that define the relationships between input features, and form representations that capture the principal components of these correlations (Oja, 1982; Linsker, 1988).

Combinatorial: This is the opposite of relational — the input is composed of a number of indi-

vidual features which do not depend on each other for the meaning of a stimulus within the task. It is somewhat difficult to come up with naturalistic examples of combinatorial stimuli in a pure sense, but many stimuli have aspects of combinatoriality. For example, numbers can be arranged combinatorially to express any quantity. However, there is some level of meaning in the left-right relationship of the numbers, so it is not purely combinatorial. Similarly, words within sentences, and letters within words, can be combined in a semi-combinatorial fashion.

The Hebbian associative learning aspect of LEABRA, given that it picks up on correlational structure, will not necessarily be of value in learning combinatorial problems, especially when the evidence for combinatoriality in the training set is not overwhelming. This is because the spurious correlations present in a given training set will be encoded by the associative learning. However, with a sufficiently large training sample, it should be the case that associative learning is helpful in accentuating the correlations that define an individual feature, while the spurious correlations will be so small as to be negligible. However, the entropy reduction imposed by the kWTA ReBel activation function should be important for learning in combinatorial domains. It encourages units to specialize on representing dissociable parts of the input, which in a combinatorial domain ends up being the independent features of the input.

Orthogonal: This is the case where there is no information in the form of overlap between different input patterns regarding the relationships between different inputs. While this case is not particularly relevant psychologically, it has been used to demonstrate that factors other than input similarity (which is not present at all in this case) can determine the structure of representations learned by neural networks (Hinton, 1986).

The effect of LEABRA on learning with orthogonal representations is studied here in the family trees problem, which was used in Hinton (1986). While it is clear that associative learning will not be of any particular use with orthogonal inputs, it turns out that in a recurrent network, associative learning can be driven by correlations across input and output patterns, not just within a given input pattern. The family trees task does exhibit this kind of input-output structure (related people tend to appear in relationships with the same subset of other people), and it is likely that associative learning will help in developing representations which capture this kind of structure.

Input-Output Mapping Structure

Categorization tasks: These tasks have a many-to-one relationship between input and output patterns, where the outputs represent groups of input patterns that all share some categorical properties. The digit recognition task, as well as a wide range of other psychologically relevant tasks, are of this type.

Given that the entropy reduction principle, which figured heavily in the design of LEABRA, is intended to encourage the development of categorical representations, it should be the case that LEABRA performs quite well on these tasks, with both the Hebbian associative learning and activity constraints being important factors.

Transformation tasks: These tasks are distinguished from the categorical tasks in that the information in the input is not collapsed into a lower-order output signal, rather it is simply transformed into a different representation in essentially an information-preserving fashion. Thus, these tasks have a one-to-one relationship between input and output patterns, which can typically be characterized according to a set of regularities and exceptions to these regularities. A special case of this is an auto-associator, where the output pattern is the same as the input pattern. Often, the input patterns for transformation tasks are somewhat combinatorial, in that transformations can be applied on individual parts of the input pattern independent of other parts. A classic example of a transformation task is the mapping between orthographic and phonological representations of words.

The role of LEABRA in transformation tasks depends on whether the inputs are combinatorial or not, as discussed above. However, in general, associative learning can be helpful in learning systematic mappings between given components to the extent that there is a strong correlation between particular input features and corresponding output features, which is typically the case.

Multiple-relation tasks: These tasks have the property of performing different mapping problems between an input and output pattern depending on the state of a relationship input pattern. A classic example of this is the family trees task (Hinton, 1986), where the network has to answer questions of the form "so-and-so's mother is who?" Thus, the network has to be capable of developing different transformational mappings depending on the content of the relationship input. Hopefully, there are shared aspects of the set of transformations over different relationships, so that learning one type of relationship benefits learning others, but this is not necessarily the case.

Given that there are many different levels of systematicity possible in multiple-relation tasks, and different levels of transfer among different relationships, it is difficult to assess the role of LEABRA in these tasks. The mere fact of having the same inputs mean something different depending on the state of other inputs (i.e., the relation inputs) is similar to the relational input type described above, which means that the relatively sparse activities in LEABRA might be an advantage.

Classification of Tasks Studied

The first task described below is the handwritten digit recognition task, which is a good example of a classification task with relational input features. Since these are cases where the features of LEABRA should clearly be of benefit, it is important to establish this first-order level of performance before moving on to more complicated cases.

In Chapter 6, the issue of generalization in fully regular domains is studied. One of the ways this issue has been examined in the literature is with a transformation task with completely combinatorial inputs (Brousse, 1993). Thus, the regularity in this task is the individual mapping between independent features in the input to their corresponding output pattern. Due to their combinatorial nature, there can be a huge number of possible input patterns, but good generalization can still be obtained after training on a relatively small sample. Also, this task represents one way of approximating psychologically important tasks like the spelling-to-sound mapping problem, which have a

Algo 1	Algo 2	Algo 2 Adds
BP	AP	Interactivity (recurrence)
AP	CHL	Weight symmetrizing (and midpoint)
CHL	ReBel + CHL	ReBel activation constraints
ReBel + CHL	LEABRA	MaxIn associative learning
ReBel + MaxIn	LEABRA	CHL error-driven learning
LEABRA	AE LEABRA	Error-driven self-structure learning

Table 5.1: Key comparisons between the different algorithms, highlighting what the second algorithm adds to the first. Thus, a comparison between the performance of these two algorithms will reveal the importance of this factor.

combinatorial, transformational nature (but not as extreme as in this case). For these reasons, this task is used as a starting point in investigating generalization. Subsequently, another fully regular transformational task is explored, this time using a combination of a relational and combinatorial input encoding. The inputs are horizontal and vertical lines on a simulated retina, so that the line is defined in relation to other active points, but each line present in the input is treated combinatorially. This task has also been the focus of previous research using different algorithms (Földiák, 1990; Saund, 1995; Zemel, 1993; Dayan & Zemel, 1995).

In Chapter 7, the issue of learning in deep networks is studied. The challenge in this case is to come up with tasks which are sufficiently complicated as to actually require additional layers of processing. One way in which this has been explored in the literature is with the family trees problem (Hinton, 1986), which is a multiple-relation task with localist orthogonal inputs, as described above. The goal of this task is essentially to develop systematic representations of the orthogonal inputs in intermediate "coding" hidden layers, making the overall task easier for a central hidden layer which takes a person input and a relationship input and outputs the correct other person. A modified version of this task with distributed input patterns with random similarity relationships is also studied.

Summary of Algorithms Used In Evaluation

A number of standard algorithms are used to compare against the performance of LEABRA on the tasks studied. In addition, comparisons with variants of LEABRA enable the contribution of the different components of LEABRA (activity constraints, Hebbian associative learning, error-driven learning) to be evaluated (see Table 5.1 for a summary). These algorithms and critical comparisons are described below, followed by a derivation of some variations of these algorithms which are used to test the validity of the implementational principles upon which the design of the LEABRA algorithm is based.

For all algorithms on all tasks, the following common set of parameters were used: individual training patterns were scored correct if all output units were on the correct side of .5, and a criterion of at least 95% correct performance over all training patterns was used for training time and to begin the scoring of generalization performance; activations were in the range between 0 and 1 (for com-

parison with LEABRA and compatibility with the constraints on the sign of neural activations); no momentum or other commonly-used learning mechanisms; on-line (per pattern) weight updating.

For each task, a BP network was trained on 5 different random subsamples of training/testing patterns, with 5 random weight initializations (networks) per each subsample. The subsamples with the best and worst mean generalization (over the 5 random networks) were then used for the other algorithms, which, due to their interactive nature, required considerably more computational resources to train. The results for the best subsample are typically presented in the figures, but results for the worst subsample were comparable, just a bit worse. Thus, there was no evidence of a subsample by algorithm interaction, meaning that the results presented on the one subsample are probably indicative of results on the task in general. Given that the critical results (comparisons between CHL and LEABRA) were (in all cases except the family trees task) massively significant, no specific statistical tests were performed. Nevertheless, the standard error of the mean (SEM), which is typically a good indicator of significance for a t test, is reported for all results.

Standard feedforward backprop (BP): This is the predominant learning algorithm used in the field. To summarize briefly, it consists of a network with feedforward-only connections, and units which compute their activation value according to the standard sigmoidal logistic function $\sigma(\eta) = 1/(1 + e^{-\eta})$ of the net input η which is the inner (dot) product of the weights times the activations. Weights are modified by backpropagating the error from the output through the hidden units, and changing the weights so as to minimize the output error. The cross-entropy formulation for the error function was always used to equate BP with the GeneRec based learning. See Chapter 2 for more details. A learning rate of .01 or .1 (the larger learning rate had mixed effects on generalization performance) was used, along with an error tolerance of .05, so that if the output activation was within .05 of the target, the unit had no error. Note that this error tolerance is unrelated to the training criterion described above.

Deterministic CHL: According to the GeneRec framework presented in Chapter 2, the contrastive Hebbian learning algorithm (CHL) used in the deterministic Boltzmann machine (DBM) is an approximation to backpropagation where the activation signals carried by recurrent weights convey error information. This is the same error-driven learning rule that is used in LEABRA, but it is implemented here in a network without activation constraints, using standard sigmoidal activation functions, etc. as originally derived. Thus, this algorithm is the primary one that LEABRA should be compared to, since it has the potential to exhibit all of the functional properties of a neocortical learning algorithm as described in the introduction, whereas the feedforward backpropagation networks can not exhibit interactive processing, and are also biologically implausible. CHL networks used net-input based averaging with a step size of .2. Settling was stopped when the maximum activation change (before multiplying by the step size) was below .01.

Almeida-Pineda Backprop: This is a version of error backpropagation which works in interactive (recurrent) networks. It is closely related to the GeneRec algorithm (see Chapter 2), but does not have a symmetry constraint on reciprocal weights, and it relies on the biologically implausible

standard form of error backpropagation. This algorithm provides an important intermediate case between backpropagation and GeneRec, since it does not tend to make use of the reciprocal weights during learning, and thus behaves more like a feedforward network than the symmetric GeneRec algorithm, while still being somewhat interactive. The same parameters as for CHL were used for step size and stopping settling.

LEABRA: This is the modal version of the algorithm, which uses the ReBel activation function, GeneRec biologically feasible error-driven learning via interactive activations, and the MaxIn Hebbian associative learning rule. The GausSig likelihood function is used by default (see Chapter 4), but results for the sigmoidal function are also reported for comparison. Some tasks show a sensitivity to this difference, while others do not. See the appendix for a discussion of the other standard parameters used.

ReBel + CHL (No As): This purely error-driven subset of LEABRA, which excludes the MaxIn associative learning component, allows the impact of MaxIn to be evaluated when compared to LEABRA, and the impact of the fixed activation constraints in ReBel when compared to the CHL algorithm.

ReBel + MaxIn (No Err): This purely self-organizing subset of LEABRA, which excludes the CHL error-driven learning component, shows how well the various *a priori* constraints implemented by MaxIn and ReBel can learn to perform tasks in the absence of specific error information. These networks will not typically be able to solve the problem to criterion, but nevertheless their learning and generalization abilities are often surprisingly good.

AE LEABRA: This is the auto-encoder (AE) version of the LEABRA algorithm, described in Chapter 4, which has error signals shaping the representation of self-structure among the input and output patterns. This should result in better such representations, and better performance relative to standard LEABRA.

AE CHL: This is an auto-encoder version of deterministic CHL, using the same idea as AE LEABRA, but obviously without the self-organizing learning.

BP and CHL with simple weight decay and weight elimination: Some variety of weight decay is the most commonly used remedy for the underconstrained behavior of backprop, and it has been reported to improve generalization. However, it typically increases training time, probably because it represents an additional adapting constraint in the error term. Both a simple weight decay (*SWD*) (proportional to the size of the weight) and the weight-elimination style weight decay (*WED*) (Weigand et al., 1991) were used. The latter has a vanishingly small weight penalty for weights that grow larger than 1, with a stronger weight penalty for smaller weights, and generally works better than simple weight decay.

BP and CHL with Noise: Neural firing is well known to be noisy, and it is possible that such noise might have beneficial effects on the ability of networks to generalize. An (1996) showed that noise in the input patterns is most likely to produce generalization benefits, so this is the form of

noise used for both BP and CHL. However, a more biologically plausible form of additive noise in the outputs of all units in the system was also tested. It is hypothesized that noise will not provide much of a benefit on the binary-valued tasks studied in this thesis.

Backprop with adapting activation constraints: This is standard feedforward backprop with versions of the adapting activation constraints described below. This will enable the evaluation of the implementational principle which favors fixed constraints (e.g., the ReBel kWTA activity constraint) as opposed to adapting ones.

Deterministic CHL with adapting activation constraints: This is CHL with versions of the adapting activation constraints described below.

An Alternative Adapting Activity Regulation Algorithm

The ReBel activation function used in LEABRA imposes a soft kWTA activation constraint on the units, resulting in graded activations with a fixed upper limit on the total activity in a layer. Thus, despite its graded nature, it is a non-adapting or fixed constraint. As discussed in the section on implementational principles in Chapter 3, it is hypothesized that this fixed constraint will result in more of the benefits of a constrained *a priori* model than will adapting constraints. In order to test this hypothesis, a couple of alternative forms of adapting activity constraints are developed here and compared against ReBel in several of the simulations reported later.

The form of adapting activity constraint developed here follows the conventional practice of introducing an additional cost term into the overall cost function that a network is optimizing. The particular implementation of this constraint follows the approach of Zemel (1993) by introducing a term in the cost function which penalizes units whose probability of being active differs from a fixed base-rate value, α. The derivative of this function is then taken, and incorporated into the learning rule.

The first version uses a cost term which computes the asymmetric divergence (cross entropy) between the unit's current activation and the desired activation α:

$$C_a = \sum_j y_j \log \frac{y_j}{\alpha} + (1 - y_j) \log \frac{1 - y_j}{1 - \alpha} \tag{5.1}$$

where the activations are interpreted as reflecting the probability of an underlying binary variable described by the Bernoulli distribution. The weights are adjusted by taking the derivative of this cost function with respect to the weight:

$$\frac{\partial C_a}{\partial w_{ij}} = \left[\log \frac{y_j}{\alpha} - \log \frac{1 - y_j}{1 - \alpha} \right] y'_j y_i \tag{5.2}$$

This can be used for both standard error-driven feedforward and the biologically feasible recurrent GeneRec networks derived in Chapter 2. For this latter case, the derivative (5.2) is evaluated using

the same approximate midpoint method as for the regular error term in order to be consistent. Thus, the sending unit activation y_i is replaced with the average of its values in the plus and minus phases.

If the GeneRec networks are viewed as deterministic or mean field approximations of Boltzmann machines (Hinton, 1989b), which they are formally equivalent to, then the above cost function should affect the activation updating procedure as well (suggested by R. S. Zemel, personal communication). Thus, the combined cost function for a mean field network with the activity constraint, where the first two terms are the standard energy and entropy of the activity states, is:

$$C = \sum_j y_j \sum_i w_{ij} y_i - \sum_j y_j \log y_j + (1 - y_j) \log 1 - y_j - \sum_j y_j \log \frac{y_j}{\alpha} + (1 - y_j) \log \frac{1 - y_j}{1 - \alpha} \quad (5.3)$$

The activation dynamics for this system can be computed by taking the derivative of this cost function with respect to the activation of unit y_j, and then setting the result equal to zero in order to determine the equation resulting in a stable equilibrium value of the activation state under this cost function. By iteratively updating y_j according to this equation, the network will settle into a stable state which is a minimum of the cost function.

$$\frac{\partial C}{\partial y_j} = \sum_i w_{ij} y_i - 2(\log y_j - \log(1 - y_j)) - (\log \alpha - \log(1 - \alpha)) \quad (5.4)$$

Setting this equal to zero and solving for y_j yields:

$$y_j = \sigma \left(\frac{1}{2} \sum_i w_{ij} y_i - \log(\alpha / (1 - \alpha)) \right) \quad (5.5)$$

where σ is the standard sigmoidal logistic function. Thus, including this constraint in the activation dynamics amounts to using a gain value of one half, and subtracting a constant bias term from the net input. Given that both of these factors can be compensated for by adapting the weights, it is likely that the weight adjustment term will be more important than this activation function. This will be tested by comparing networks with just the weight derivative term (5.2) against networks with both the weight derivative and the modified activation dynamics given by (5.5).

Finally, I also explore a variant of the above approach which might avoid the pressure to keep all activations around α activation level with little variance across patterns. This variant simply substitutes a running-average estimate of the unit's mean activity level for the instantaneous activity of the unit. This allows the unit to have variation around the mean on particular cases without being penalized. The activity cost term is thus:

$$C_a = \sum_j \overline{y_j} \log \frac{\overline{y_j}}{\alpha} + (1 - \overline{y_j}) \log \frac{1 - \overline{y_j}}{1 - \alpha} \quad (5.6)$$

where $\overline{y_j}$ is a running average which is updated by:

$$\overline{y_j} = \epsilon y_j + (1.0 - \epsilon)\overline{y_j} \tag{5.7}$$

While these forms of adapting activity constraints do not place exactly the same kinds of constraints on the units as the ReBel function used in LEABRA, they have been used by other researchers, and exemplify the simple approaches typical of such mechanisms. A constraint that would be closer to ReBel would involve the use of an instantaneous average value computed across the layer at a given point in time. However, such a value is no longer local to the individual unit, and thus could not simply be substituted into equation (5.6). A more complicated mechanism would need to be introduced in order to express the proper relationship between an individual unit and its contribution to this instantaneous average. While this is certainly possible to do, it raises a number of additional implementational issues, and would only be justified if the adapting constraints described above showed some promise. However, they do not appear to work well at all in practice, as will be discussed below.

Failure to Successfully Implement a Fully Adaptive form of LEABRA

It is possible that an alternative version of central ideas behind LEABRA could be formulated by combining the adapting activity constraints described above with a standard associative learning rule like the soft competitive learning (SCL) rule (Nowlan, 1990). This alternative version would correspond to a more "standard" way of implementing LEABRA, and would provide a useful comparison for evaluating the relative advantages and disadvantages of the particular implementational choices made in the version of LEABRA described in this thesis. However, I found that it was not possible to successfully implement such a system, for the following reasons.

The primary reason for failure is that the SCL associative learning function relies on the activation constraint to determine which units become associated with the current input. When they do become associated, their weights move towards the values of the input patterns. Thus, with an adapting constraint, too many units are allowed to become active, and they end up adapting their weights towards a mush of many different input patterns, resulting in useless, non-selective representations. Further, the activation dynamics of such a system do not work well when the weights are encouraged to all be positive, since this requires a delicate balancing of this excitation with a negative bias. Building in these parameters from the start helps, but the system is still not stable over time and, with extensive parameter searching, I was not able to get it to learn even simple problems.

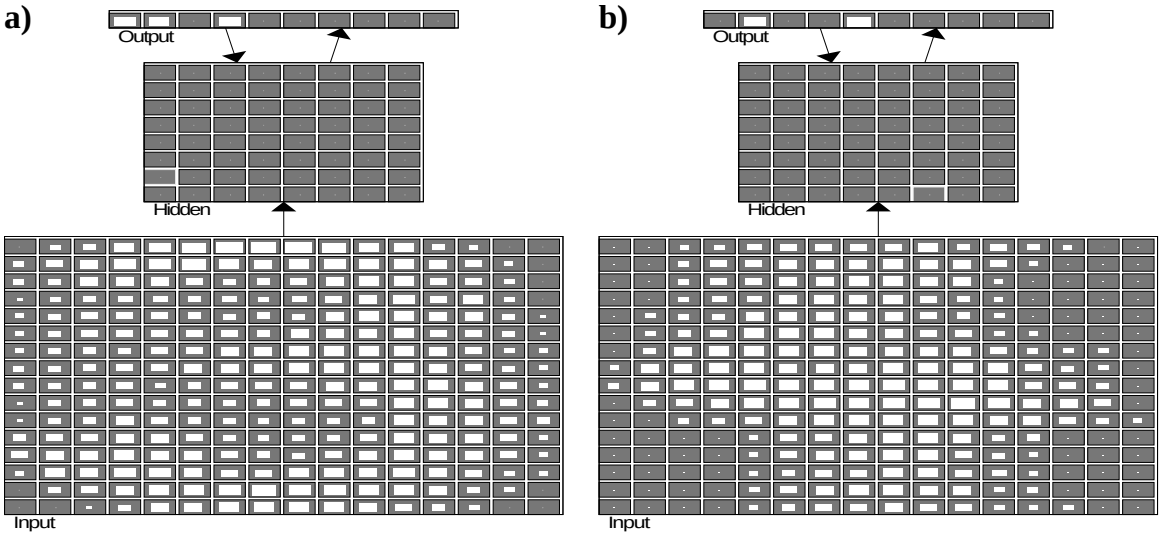

Figure 5.1: Weights for two different hidden units in the digit task for a LEABRA network (both input-to-hidden and output-to-hidden weights are indicated in the corresponding input or output unit, with the area indicating magnitude and color indicating sign (white = positive, black = negative)). a) Shows a unit that participated in the representation of the digits 0, 1 and 3. b) Shows a unit which participates in the representation of digits 1 and 4. The weights clearly reflect important properties of the visual appearance of these digits.

A Test Case: The Digit Recognition Task

The handwritten digit recognition task has been studied widely in the neural network field, and it represents a good example of a difficult categorization task. The different digits are represented by overlapping patterns over the same set of input units, so they are relational inputs as described above. A given digit can be drawn in many different ways, and can appear in different positions on the input. The particular digit set used in this work is the same as the one used in Nowlan (1990), which contains 48 versions of each of the ten digits (0-9) in a 16x16 pixel format. 64 hidden units were used for the results reported below, and the LEABRA networks had a k activity parameter of 16 units active, resulting in 25% activity levels. Instead of using just 1 unit per output class (digit), 3 redundant units were used in the LEABRA network for reasons that are described below (this manipulation did not affect results for the other networks, which thus used only 1 for computational efficacy). Unless otherwise noted, the associative learning strength for LEABRA was .5. Other parameters were standard as described above and in the appendix.

As with most classification tasks, this one has a high dimensional input space and a relatively low dimensional output. Thus, there are probably a large number of different ways to successfully carve up the input space so as to get good performance on the training set. As a result, this problem does not strongly constrain the units to a particular solution, as is typical of most problems, and must be the case with neurons in the neocortex, which have tremendous excess degrees of freedom given the number of neurons and synapses. It is therefore expected that the purely error-driven algorithms (BP, AP, CHL) will develop under-constrained, noisy representations, and that the interactivity in

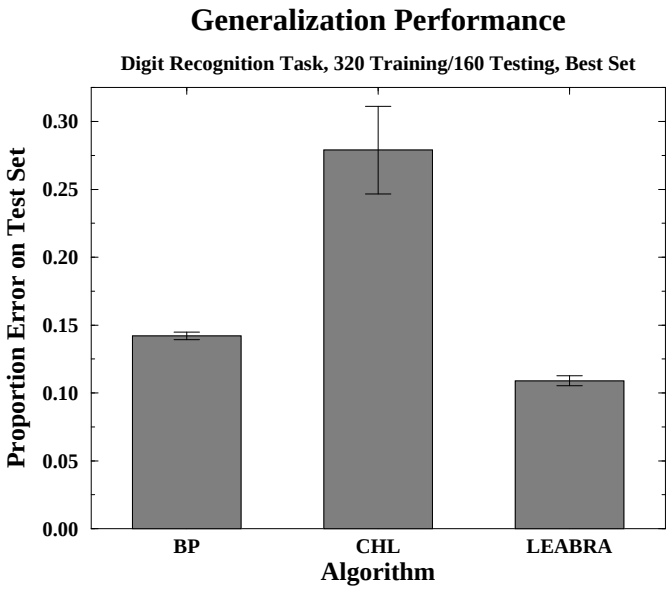

Figure 5.2: Average best generalization performance (proportion of error on the testing items, N=5) in the digit recognition task for BP, CHL, and LEABRA. LEABRA shows the best overall performance of the three. CHL is significantly worse than both BP and LEABRA, which is consistent with the idea that the under-constrained, error-driven weights cause the interactive CHL network to treat similar inputs differently.

CHL will result in poor generalization. In contrast, the entropy reduction biases built into LEABRA should result in representations which extract the correlational structure of the inputs, and thus have a smoother activation space and better generalization.

Thus, this task represents a basic test case for the central ideas behind the LEABRA algorithm. Fortunately, the results are consistent with the theory. This can be seen by examining the types of representations formed by LEABRA, and by its generalization performance compared to other algorithms. Figure 5.1 shows two representative cases of hidden unit weights for LEABRA in this task. There are several important properties of these weight patterns. One is that they clearly reflect important properties of the visual appearance of the digits they represent, which means that the units have captured the structure of this environment. This can be compared to the results for the feedforward backpropagation network, which was shown in Figure 1.2 to have weight patterns after training that are difficult to distinguish from the random initial weights.

Another important property of the LEABRA weights is that the hidden units clearly have a sparse distributed code for the digits (each unit participates in the representation of from 1 to 3 digits), and that this code appears (based simply on visual inspection of many hidden units) to reflect the similarity (pattern overlap) structure of the digits. In comparison, the distributed representations for digits in the purely error-driven networks are considerably more diffuse, with many units participating in the representation of all the digits, and they are much more random with respect to the perceived visual similarity of the digits (again, based on visual inspection).

While the above properties of the hidden unit weight patterns developed by LEABRA are im-

portant in their own right, it is also important to demonstrate that they have beneficial functional consequences. Thus, the generalization performance of these different algorithms was examined by training on a subset of the data (32 samples of each digit) and testing on the remainder (16 samples per digit). Five different random sets of the training and testing items were generated, and the BP algorithm was run on all of them with standard parameters (see above). The set that resulted in the best generalization performance was then used for evaluating the other algorithms, which all require significantly more computation (and thus training time) than BP. Generalization performance was measured over 5 training runs of 200 epochs each, with testing occurring after every 10 epochs. All generalization performance scores are reported as proportion *error* (not correct) out of the total number of testing items, and testing performance was measured only after the network achieved the standard training criterion of 95% correct (no more than 16 errors out of 320 patterns). The average of the peak performances for each of the 5 runs is reported except where noted otherwise. Note that unlike a feedforward error-driven algorithm, LEABRA exhibits a certain amount of fluctuation in performance after reaching the training criterion, probably due to the constant balancing of error-driven and associative learning and the fact that it is a relatively sensitive, interactive network. Deterministic CHL networks exhibit this fluctuation as well, probably due to their sensitivity to small weight changes over learning. Thus, the average generalization scores over training are generally a bit worse than the peak scores for these algorithms.

Figure 5.2 shows the generalization performance for the BP, CHL, and LEABRA algorithms. The interactive CHL network generalizes significantly worse than the feedforward BP network, which is consistent with the hypothesis that interactivity and under-constrained representations do not lead to good generalization. In contrast, the constrained, structured representations developed by LEABRA result in the best generalization performance, despite the fact that LEABRA is an interactive network. However, the fact that LEABRA appears to perform slightly better than BP in this case, while interesting, is not of central importance. The critical result is that LEABRA generalizes substantially better than CHL.

Figure 5.3 shows results that reveal the importance of the different components of the LEABRA algorithm. First, the ReBel + CHL condition (*No As*) can be compared with the standard CHL network to see the effects of the ReBel activity constraints, which is the difference between these two cases. Clearly, ReBel provides a significant improvement in generalization performance. This can be attributed in part to the activity constraint making each unit more responsible for representing particular input patterns. In addition, the ReBel constraints provide a damping effect on the activation dynamics, so that the network is less sensitive to differences in input patterns. This will be explored further in Chapter 6.

The contribution of the MaxIn associative learning can be seen by manipulating the strength of this component (relative to the error-driven CHL term), indicated by $as = .1, as = .5, as = 1.0$ in the figure. As MaxIn plays a greater role in learning, generalization improves, with an apparent ceiling effect above the .5 value. The importance of MaxIn is also supported by a visual inspection

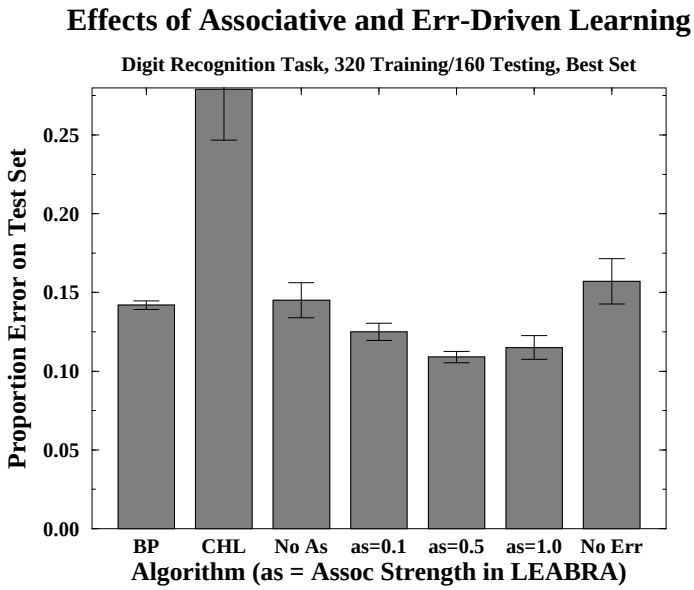

Figure 5.3: Effect of MaxIn associative learning on generalization performance (BP and CHL included for reference). **No As** (ReBel + CHL, no MaxIn) shows purely error-driven version of LEABRA, and, compared to CHL, shows significant advantages of the ReBel activation constraints. **as=.1 - 1.0** shows that as the strength of the associative learning component is increased, generalization performance improves, with maximal performance occurring with a strength of .5, and staying roughly constant up to a level of 1. **No Err** (ReBel + MaxIn, no CHL) shows purely associative version of LEABRA, which performed slightly worse than purely error-driven, but was not capable of learning the task to criterion in the first place (not shown). Thus, both associative and error-driven learning work together to give the best generalization performance.

of the weights, which looked more noisy and like those of the purely error-driven BP and CHL networks for the *No As* case (not shown), compared to the smooth representations shown in Figure 5.1. However, the final data point (*No Err*) in Figure 5.3 shows that associative learning alone is not sufficient. This shows the purely associative ReBel + MaxIn case, with no error-driven learning, which generalizes worse than even the purely error-driven version of LEABRA. This is true despite the fact that increasing the strength of associative learning (in the context of error-driven learning) results in significantly better generalization. Further, the MaxIn-only networks never learned the problem very well in the first place, failing to reach the 95% training criterion (a criterion of 80% was used instead, and even then one network failed to achieve that level of performance). Thus, it is apparent that the good generalization performance of LEABRA depends critically on the interaction of both error driven and associative learning together.

Almeida-Pineda vs *CHL Networks*

The principal theory as to why the CHL networks generalize worse than their feedforward BP counterparts is that their interactive activation dynamics make them highly sensitive to differences in input patterns. While a more explicit test of this idea will be performed in the next chapter, it is possible to use the comparison between Almeida-Pineda (AP) and CHL to establish some evidence in support of this theory. This is because the AP networks do not end up using the reciprocal (top-

Algorithm	Initial Weights	Trained Weights	Cycles	Gen Err	SEM
BP	—	—	—	.142	.00274
AP	.253	.285	48	.202	.0126
CHL	.244	.512	70	.279	.0323

Table 5.2: Comparison between measures of interactivity and generalization in AP and CHL. Weights are average absolute value of reciprocal (top-down) weights from output to hidden units in Almeida-Pineda (AP) and CHL networks. The CHL network clearly increases the strength of these weights over training, while the AP network does not. CHL networks take significantly longer to settle during processing, as is shown in the Cycles column (initial settling time before training was 40 cycles). The level of interactivity between BP (none), AP, and CHL is consistent with their respective generalization error scores.

down) weights from the output to the hidden units to solve problems (indeed, Chapter 7 shows that these networks are actually incapable of learning a particular task that specifically requires interactive processing). Since the reciprocal weights are not strongly developed in AP networks, they behave more like feedforward networks than the CHL networks, which develop substantial reciprocal weights due to the symmetry constraint as discussed in Chapter 2.

The results shown in Table 5.2 are consistent with these predictions. The average magnitude of the reciprocal weights after training is nearly twice as large for the CHL networks as the AP networks. The magnitude of these weights in the AP network is essentially unchanged from the initial state, indicating that the AP network does not use them during training. Further, the CHL networks took significantly longer to settle during processing of items after training than the AP networks, indicating a greater amount of interactivity. Initial settling time for both networks was 40 cycles, so the AP networks did not substantially increase their settling time over training, while the CHL networks did. Finally, these measures of interactivity are consistent with the observed levels of generalization, with AP performing intermediate between CHL and BP (which is not interactive at all).

Auto-encoder (AE) LEABRA

The MaxIn associative learning in LEABRA causes hidden units to represent the correlational structure of the input and output patterns. The representation of this self-structure is important for generalization, since it causes the weights to have a smooth, center-of-mass or prototype representation of the digits they encode. The use of the auto-encoder (AE) form of LEABRA results in the introduction of error signals into the learning of representations of this self-structure in the input and output patterns. This should improve the quality of these representations over those developed purely by associative learning in the standard version of LEABRA, since they will more closely reflect the important distinctions between different digits, as a result of having to reproduce the input and output patterns. An important comparison simulation is one where, instead of reconstructing the input pattern over the input layer, as is done in AE LEABRA, the input is reconstructed over an additional output layer, as is done in a standard auto-encoder. It is hypothesized that AE LEABRA is beneficial because it provides error signals for training the same weights which are used to perform the input-output mapping task, which is only true if the input is reconstructed over the same input

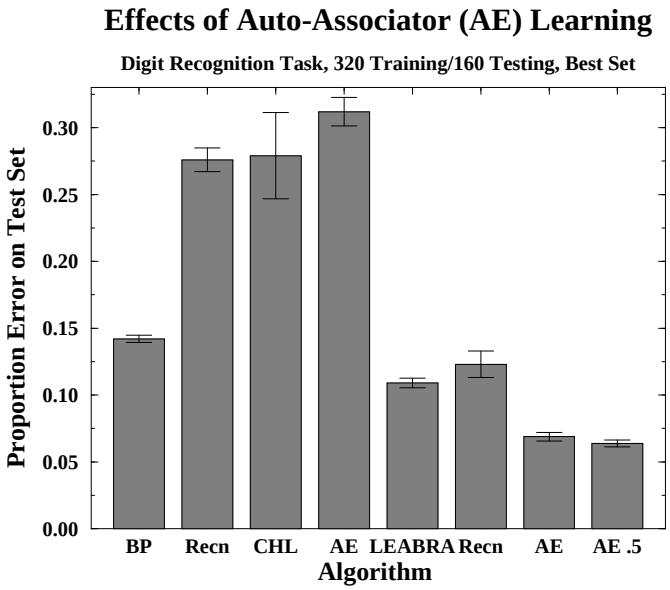

Figure 5.4: Comparative generalization performance for auto-associator (AE) versions of CHL and LEABRA, with comparison to output-layer reconstruction (Recn) versions. **AE .5** is AE LEABRA with .5 decay of plus-phase activity state prior to the reconstruction phase. While the auto-associator learning clearly improves LEABRA's performance significantly, and this improvement is slightly better with the activity decay, it appears to impair performance in CHL. The Recn task, which uses a different set of weights to a separate reconstruction output layer for solving the auto-encoder problem, impaired performance significantly in BP, and somewhat in LEABRA.

layer. Thus, this output-layer reconstruction task provides an important control for any benefits that might be derived simply from the effects of the reconstruction task on the hidden unit representations themselves, apart from the effects on the input-to-hidden weights.

Figure 5.4 shows that AE LEABRA exhibits substantially improved generalization performance over standard LEABRA in this task, which is consistent with the idea that the representations of self-structure are cleaner and more closely aligned with the actual structure of environment. Interestingly, the use of pure error signals in the auto-encoder version of CHL did not improve generalization performance (actually, it appeared to slightly impair performance). Thus, there is some indication that the auto-associator error signals are only helpful in the context of associative learning on this task. The results on the output-layer reconstruction task (*Recn* in the figure) for both BP and LEABRA show an impairment (a significant one in the case of BP), indicating that the advantages of AE LEABRA are in its ability to provide error signals to the input-to-hidden weights, and not in the effects of reconstruction task itself on the hidden representations.

To evaluate the additional contribution of associative learning in AE LEABRA, the ReBel + CHL condition was run in the auto-encoder version, so that only error signals were driving learning of both the mapping and self-structure. While average peak generalization scores in this case were reasonably good (.09, SEM .00719, compared to .0688, SEM .00312 for the case with associative learning strength of .5 reported in the figure), this peak generalization was very early in training, and gener-

alization declined as it proceeded. If only the final (after 200 epochs) generalization scores are measured, performance deteriorates considerably (.224, SEM .0387). In contrast, no such trends over training were apparent in standard (non auto-encoder) networks, or in AE LEABRA with associative learning (which had, for comparison, .0862, SEM .00559 generalization error at 200 epochs).

Thus, there appears to be a significant *overfitting* problem associated with the auto-associative pressure to represent the input data, which, in this case, can be considered to be quite noisy, given that there is a good deal of individual variation in the way the digits were drawn. The MaxIn associative learning is presumably capable of preventing this overfitting by aligning the representations with the central tendency over all instances of the subset of digits represented by a given unit. Additional data in support of this argument comes from the error levels associated with the reconstruction of the input and output patterns. For the case with .5 MaxIn associative learning, the reconstruction mean-squared-error (MSE) per unit was approx .085, while it was approx .061 for the case without MaxIn. Thus, the associative learning results in poorer reconstruction of the inputs, indicating that the representations have captured less of the details, but the correlational structure captured by the associative learning is evidently important for generalizing to novel inputs.

These results provide additional evidence that error-driven and associative learning in LEABRA work better together than alone. Further, the substantial improvement in generalization performance associated with the use of auto-encoder learning in LEABRA indicates that this could be of important practical interest. It should be noted that even small improvements over already low generalization error scores are highly significant due to the increased difficulty of extracting such improvements. Finally, it is interesting to note that, because AE LEABRA requires interactive processing in order to reconstruct the input pattern over the input layer itself (which was shown above to be essential for its success), this turns interactivity into an important contributor to impressively good generalization, as opposed to the obstacle it is in a standard CHL network.

LEABRA vs *Weight Decay*

Since weight decay is perhaps the most commonly-used means of imposing constraints on error backpropagation learning, its effect on generalization performance was assessed on this task. Figure 5.5 shows the results compared to LEABRA and standard backpropagation and CHL with and without weight decay. As shown, a small amount of weight-elimination (WED) weight decay (.001) does not make much of a difference, while an intermediate amount (.002) results in slightly better generalization, but a larger amount (.005) results in worse generalization than without any weight decay. Thus, the effects of weight decay are parameter sensitive, and too much weight decay is a bad thing. Compare this to the results for increased associative learning strength in LEABRA, which was considerably more robust (a strength of 1 has comparable performance to that of .5). Finally, the LEABRA network outperformed the BP network with .002 WED, indicating that weight decay is not as effective as the associative learning and activity constraints built into LEABRA on this task.

While weight-elimination weight decay in the BP and CHL networks with 64 hidden units did

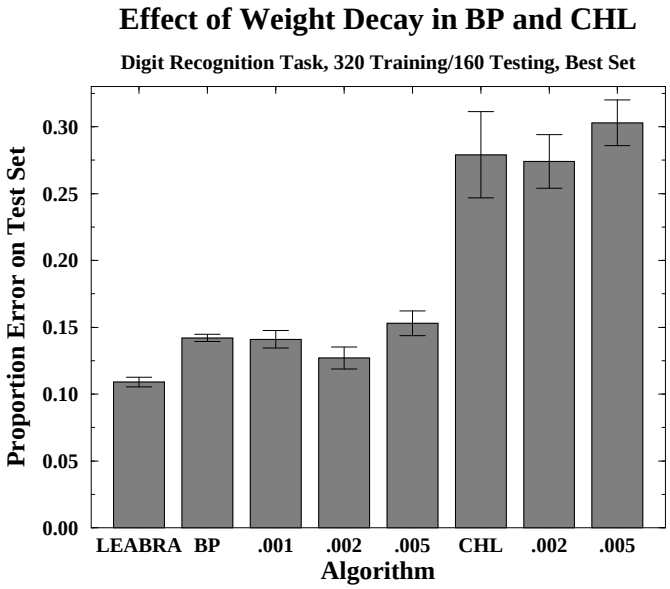

Figure 5.5: Comparative generalization performance for the weight-elimination version of weight decay (WD) for both BP and CHL. The benefits of weight decay are parameter sensitive (.001 barely makes a difference, .002 gives slightly better performance, but .005 results in worse performance) and do not match those of LEABRA.

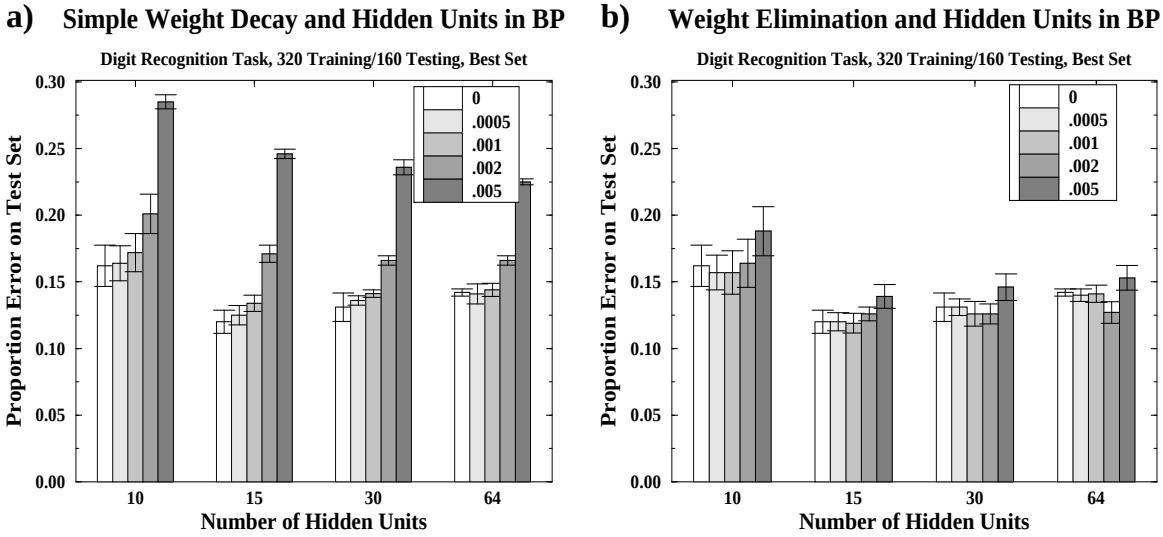

Figure 5.6: Results of search of hidden unit and weight decay parameter space. **a)** Results for simple weight decay at five different levels (including 0) and four different numbers of hidden units. **b)** Results for weight elimination decay at the same levels and hidden units numbers.

not reach the generalization performance of LEABRA, it is possible that simple weight decay (SWD) and/or fewer hidden units might result in better generalization. Thus, the space of hidden unit size, weight decay type, and weight decay level was searched for BP networks using a simple grid of parameters. 10, 15, 30, and 64 hidden units were used, with both WED and SWD at levels of 0, .0005, .001, .002, and .005. The results are shown in Figure 5.6. The use of SWD did not result in better generalization for any of the cases relative to no decay at all, but WED did show a slight improvement, which was most substantial at the .002 level in the network with 64 hidden units. A slight generalization advantage was observed for smaller numbers of hidden units down to 15, but not with only 10. This is consistent with the idea that there are fewer free parameters in networks with smaller numbers of hidden units, and thus the networks are more constrained by the training data. However, the magnitude of this effect appears to be small. Further, the reliance on tightly controlled numbers of hidden units is not biologically plausible, since the cortex has such a huge number of neurons.

While some combinations of weight decay and number of hidden units might result in statistically comparable generalization performance in BP and LEABRA, the optimal version of BP probably does not generalize *better* than LEABRA, and almost certainly does not approach the performance of AE LEABRA. However, the critical point is the comparison between CHL and LEABRA, not BP and LEABRA. While it was too computationally expensive to search the parameter space of hidden units and weight decay in CHL, it is likely that small effects observed in BP apply to CHL as well. As a simple test of this, CHL networks with 15 and 30 hidden units were run. Generalization was only slightly improved over the 64 hidden unit case: 15 hu, .236, SEM .00809; 30 hu, .231, SEM .0106 compared to .279, SEM .0323 for 64 units reported above, but not enough to bring performance even close to the range of the other algorithms. When WED .002 weight decay was added, the 15 and 30 hidden unit results were actually worse than CHL without decay: 15 hu, .262, SEM .011; 30 hu, .234, SEM .0132.

LEABRA vs *Noise*

Training with noise is commonly thought to serve as a means of regularizing or constraining a network in a manner similar to that of weight decay, but more closely resembling the use of a smoothness constraint (Poggio et al., 1985). An (1996) recently showed analytically that some ways of applying noise provide a useful regularizing function, and others do not. In particular, noise added to the input vectors was shown to impose a smoothness constraint on the function learned by the network. However, this appears to be applicable largely in cases where the inputs have continuous values, so that the noise effectively samples over a larger continuous space. In cases where the inputs are binary, as in this task, noise is likely to be substantially less useful, since the important dimensions of sampling are over different binary patterns across units, and not over continuous values of a given unit's activity. It is likely that in the brain, neural firing is not precise enough to be encoding continuous values, so that this binary case is more relevant. As can be seen in Figure 5.7, noise added

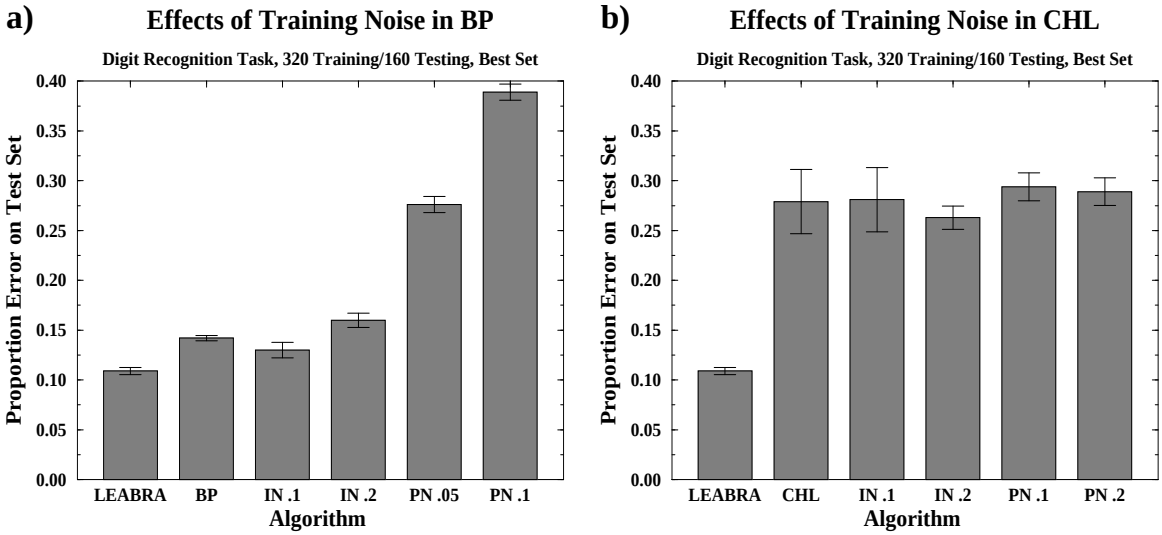

Figure 5.7: Comparative generalization performance for the use of noise during learning for both BP and CHL (IN is input noise, PN is processing noise (added to all units), .05, .1 and .2 are the variances of the Gaussian, 0 mean noise). The benefits of noise are slight, parameter sensitive, and depend on the algoirthm. In no case does it match the performance of LEABRA. Note that PN as low as .1 prevented the BP nets from learning the problem to criterion.

either to the inputs (input noise, IN) or to all units (processing noise, PN) did not improve generalization reliably or substantially at any of the tested levels. Furthermore, even .1 variance processing noise added to the backprop networks prevented these from learning the task to criterion. The CHL networks appeared to be more robust to the effects of noise during learning, since they were able to learn with even .2 variance processing noise. Larger amounts of noise (.5) prevented all networks from learning.

Fixed vs *Adapting Activity Regulation*

As discussed above, the purely error-driven ReBel + CHL (*No As*) case shown in Figure 5.3 shows that the ReBel activity constraints result in a significant improvement in generalization performance over a standard CHL network. It is of interest to determine if similar such improvements can be obtained by using the adapting activity constraints described earlier in this chapter. The prediction is that they will not result in as substantial of improvements since they do not constitute a true *a priori* constraint, and must trade-off against error reduction.

Figure 5.8 shows a comparison between all of the activity regulation algorithms on generalization performance. The final generalization performance over all epochs is shown in addition to the average peak ("best") generalization to illustrate the problems with the version of the adapting activity constraints based on the instantaneous activity value. In this case, generalization performance deteriorates significantly after the maximum generalization performance is reached, resulting in the bad final score shown in the figure. For most of the adapting cases, a cost weighting factor of .05 was used, which resulted in average activity levels over the hidden layer within .02 of the target .25

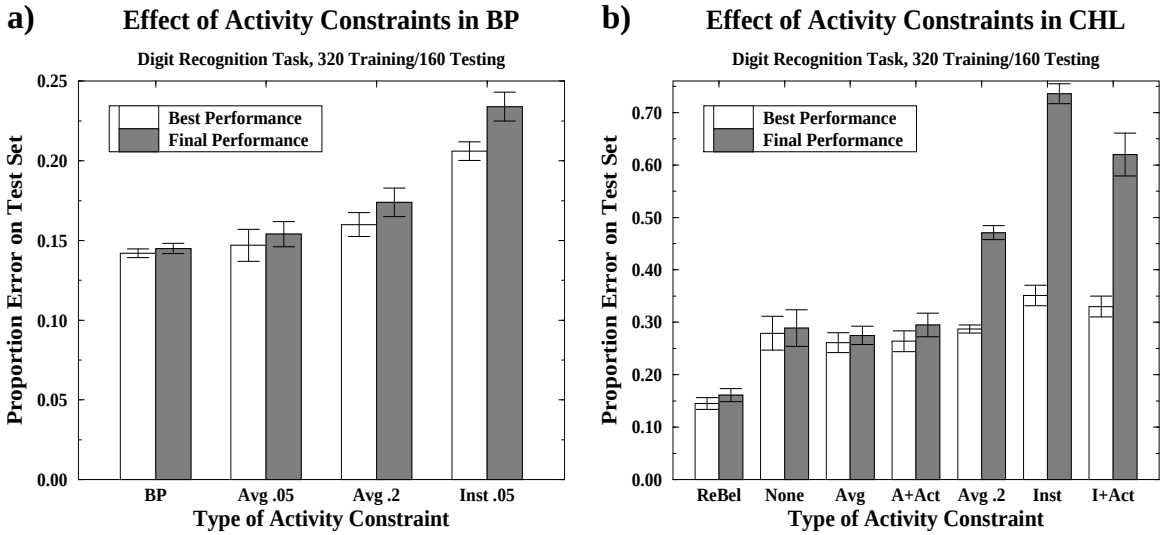

Figure 5.8: Generalization performance (*Best* is average peak performance, *Final* is average final performance) for adapting activity regulation in the **a)** BP and **b)** CHL algorithms. All act cost factors were .05 unless otherwise noted. **Avg** uses time-averaged activations in the learning cost function. **Avg .2** is Avg with cost factor of .2. **A+Act** is Avg with activation updates using the cost. **Inst** uses instantaneous activation. **I+Act** is Inst with activation updates using the cost. **ReBel** is the fixed kWTA activity constraint used LEABRA (using only CHL error driven learning, No As). **None** is a standard CHL network. There is a clear advantage for the fixed ReBel constraints in CHL. Among adapting constraints, the +Act condition makes essentially no difference, and the Inst case deteriorates over epochs of training, resulting bad final performance. Increasing the cost factor results in worse performance.

activity level (the same activity level used in the ReBel case). Smaller values did not enforce the activity constraint very well and had little impact (not shown). Larger values led to a deterioration of performance, as is shown by the *Avg .2* case in the figure.

There are two important points that can be drawn from these results. First, as was already noted, it is clear that at least the ReBel form of activity constraints can result in improved generalization performance (relative to no activity constraints) on this task. This is likely due to the entropy reduction and damping effects, which should facilitate performance on categorization tasks such as this one. Second, it is clear that the adapting activity constraints are substantially less effective than the fixed constraints provided by ReBel in this task. There was only a marginal benefit of the average-activation based constraints, and a clear disadvantage of the instantaneous-activation based constraints. One important difference between the fixed and adapting constraints is that the adapting constraints do not directly affect the activation dynamics of the network, whereas the fixed constraints have a damping effect as described previously.

Other Parameters Affecting Generalization Performance

There are several parameters of LEABRA and the other algorithms that have an effect on the generalization performance. For BP, using a fast learning rate (.1) resulted in worse generalization (.189, SEM .00809) compared to the slower one (.01) reported above.

Pct Activity	Gen Err	SEM
12.5	.138	.00494
18.75	.129	.00815
25	.109	.00356
31.25	.116	.00844
37.5	.122	.00609

Table 5.3: Effect of activity level on generalization performance in LEABRA. The 25% case is what was reported above.

For LEABRA, the 25% activity (16 units) and 64 hidden unit case worked the best, resulting in the .109, SEM .00356 generalization error reported above for associative learning strength of .5. Smaller numbers of hidden units (49 units, generalization error .123, SEM .0028) had a measurable effect, as did variations in the activity level, as shown in table 5.3. Sparser activity levels result in worse generalization than higher activity levels, which is consistent with the idea that sparse activity leads to highly specific, conjunctive representations that do not generalize well (O'Reilly & McClelland, 1994).

The use of GausSig *vs* sigmoidal units in LEABRA (see end of Chapter 4 for a discussion) was not critical for this task (sigmoidal units had a .111, SEM .00559 generalization error), though a k_{gauss} parameter of 1 for GausSig worked better (the results reported above use this) than the maximal value of 2 (generalization error .124, SEM .00746). This is probably due to the high dimensionality of the input space, which makes patterns generally further away from each other, requiring less of the Gaussian distance penalty. The standard practice (see appendix) of having a k_{gauss} parameter of 0 for the connections from the output layer to the hidden layer (since the output layer has local representations) was used (otherwise the generalization error is .147, SEM .00356).

It is possible to manipulate the relative contribution of the two terms that make up the MaxIn associative learning rule, in order to determine their relative importance. These terms correspond to a zero-sum Hebbian (ZSH) learning rule, which performs entropy reduction, and a soft competitive learning (SCL), which performs performs information preservation (see Chapter 4 for a discussion). The standard weighting of these two terms is that SCL is .25 of the ZSH magnitude, which itself is weighted dynamically in proportion to the complement of the signal-to-noise ratio of the unit (as measured by a likelihood ratio). Given that this dynamic weighting averages around .5, the SCL term is really only half as influential as ZSH on average. If a smaller (.1) or larger value (.5) of the SCL weighting factor is used, generalization performance decreases: $k_{scl} = .1$ has a generalization error of .126, SEM .00677; $k_{scl} = .5$ is .11, SEM .00523. Thus, while larger SCL value does not result in significantly worse performance than the standard .25 (.109, SEM .00356) in this task, the same is not true of later tasks. This indicates that both terms of MaxIn are important for good performance.

There is an interesting architectural manipulation that makes a measurable difference on generalization, which is the use of three redundant output units for each digit in the output layer, instead of a single unit (generalization error is .126, SEM .00407 for the single unit case compared to .109, SEM

.00356 for the three unit case). This redundancy is important because of the positive-feedback nature of associative learning, which will tend to capitalize on the initial random patterns of weights for forming associations. By increasing the sample size of these random weights, the dominance of this initial random bias is minimized. Further, each of the three weights could be used to nurture an association, improving the chances that useful (e.g., in terms of reducing error) associations are formed. This manipulation made no difference when tested in BP networks (.14, SEM .00356 generalization error), as would be expected since the advantages are related to the use of associative learning. As mentioned earlier, all LEABRA results reported above are for the case with three output units per digit.

Discussion

To summarize, the results on this task support all of the major hypotheses regarding the performance of the LEABRA algorithm on a categorization task with relational inputs. Critically, it was shown that both error-driven and associative learning together performed better than either alone, and that activity constraints were important even in a purely error-driven context. Further, the implementational principle which favors the use of fixed instead of adapting constraints was supported by the failure of the adapting constraints to perform as well as the fixed activity constraints imposed by ReBel. The auto-associator version of LEABRA performed substantially better than any other algorithm tested here, and shows potential for practical use. None of the standard regularizers tested (weight decay and training with noise) were capable of performing as well as LEABRA with associative learning. The next chapter investigates the underlying causes of generalization performance in greater depth.

Chapter 6

Generalization: Within and Across Tasks

This chapter explores the issue of generalization, or systematic responses to novel stimuli, in some detail. For generalization to occur, it must be the case that there is an underlying regularity in the environment, so that novel stimuli can be related to those the network has seen before, and treated appropriately. It is clear that the ability to generalize is an important property of neocortical processing, as there are many examples of it in the behavior of humans and animals. For example, it is possible for people to pronounce strings of letters that they have never seen before, as long as these strings follow some basic regularities. Further, people can make spatial and other judgments on novel visual displays depicting "possible" objects in the world, but they do not perform as well for objects that do not adhere to the regularities that define possible objects. These behaviors can be explained by assuming that the neocortical learning system can develop representations which capture the regularities of these domains, and thus treat novel stimuli systematically according to these regularities.

Despite the intuitive appeal of the above description of generalization in the neocortex, it can be quite difficult to actually specify what counts as a "regularity" in the environment, and how neural representations could be structured so as to capitalize on these regularities. In the previous chapter, a categorization task was explored, where the definition of regularity was adherence to a visual form of a digit, and it was easy to literally see that the representations in LEABRA reflected that regularity. Indeed, many categorization tasks have a natural definition of generalization, which is simply the ability to collapse an appropriate range of variability around a prototype or central tendency into an existing, known representation. The results from the previous chapter on the digit recognition problem support the idea that the entropy reduction constraints built into LEABRA facilitate this form of generalization.

However, there are many other types of regularities other than those of the form present in categorization tasks. Using the taxonomic distinctions made in the previous chapter, it should be the

case that both transformational and relational tasks can exhibit regularities sufficient to allow generalization. A particularly clear and simple case is that of a transformational task with combinatorial inputs, where the regularity is in learning that each of the independent features of the input is associated systematically with a corresponding feature in the output, but not with any of the other independent features. However, this is not a particularly interesting form of regularity, since the meaning of a stimulus is typically defined by the relationship among its features, or in relation to other things. In a domain where there is no relationship between features of the input, there is no real meaning to the stimulus as a whole. Nevertheless, it seems plausible to assume that some aspects of the world are like this, at least at some level. Indeed, as mentioned previously, the spelling-to-sound mapping is at least somewhat combinatorial.

The issue of generalization in a transformational task using combinatorial inputs was studied by Brousse (1993) for feedforward backpropagation networks, with and without weight decay. It was found that backpropagation networks did not develop separate representations for each of the independent "slots" in the input, except in the case with weight decay. Nevertheless, BP networks without weight decay generalized surprisingly well, despite there being no clear relationship between their weights and the independence of the different input features. While Brousse (1993) did not study an interactive error-driven network, it is hypothesized that such a network would perform considerably worse than the feedforward BP networks on this task. Further, it is hypothesized that a LEABRA network would develop representations that reflect the independent input features, and generalize well as a result. These hypotheses are tested in the first section of this chapter using two versions of the transformational task with combinatorial inputs.

The next section in this chapter explores the case of a transformational task which uses stimuli that are both combinatorial and relational. This is the "lines" task, which has been studied by a number of researchers (Földiák, 1990; Saund, 1995; Zemel, 1993; Dayan & Zemel, 1995). The stimuli are relational because a given line is defined by the relationship between a number of active "pixels" in an simulated visual input, but they are also combinatorial because a number of different lines are present at the same time, and each is treated independently of the other. Thus, this task is used to explore the basic phenomenon of generalization under different and somewhat more interesting circumstances than the purely combinatorial domain.

Finally, the issue of generalization *across* tasks is studied using both the digit classification task from the previous chapter, and the lines task just described. This form of generalization, also known as transfer, is important because it demonstrates the flexibility of the knowledge learned in solving a given task. If a limited amount of learning experience in a network model can be effectively applied to performing novel tasks, this might help to explain the basis of similar abilities demonstrated by humans. It is hypothesized that the ability of LEABRA to represent the structure of a given domain will be important for exhibiting good across-task generalization.

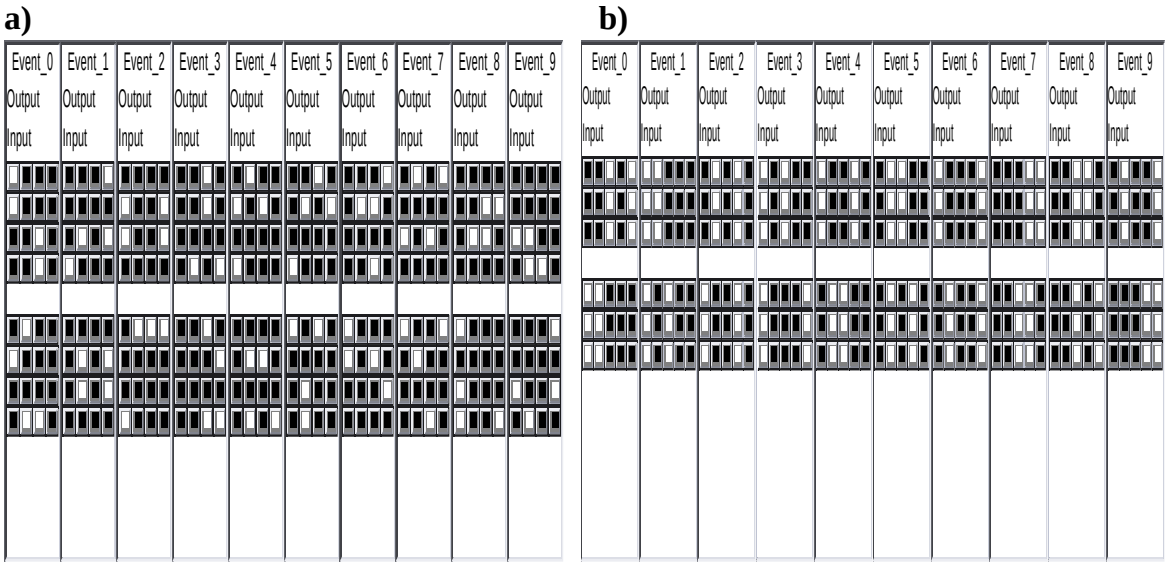

Figure 6.1: Patterns for the individual features in the combinatorial domain. a) Shows the random, relatively non-overlapping patterns used for the first version. b) Shows the 50% overlapping patterns used for the second version.

Generalization in a Combinatorial Domain

Two versions of a transformational task with combinatorial inputs were used. Both versions had four different independent input features or "slots", each of which could take on 10 different values. Thus, there were 10,000 different possible input patterns (10^4). For the first version, each of the ten feature values were represented as random bit patterns over 16 units, with four bits active out of the 16. The maximum overlap between any two patterns was 1 active bit, resulting in a simple mapping problem that should be easy for networks to learn. The transformational mapping was a random pairing of the 10 bit patterns with each other. Thus, bit pattern 1 always predicted a corresponding bit pattern of 6 on the output slot, for example. Figure 6.1a shows the patterns for this version of the task. The second version was similar, except that the bit patterns were systematically designed to have 50% overlap with each other, making the feature mapping problem more difficult to solve. This was accomplished by turning on two bits out of 5 in all the unique combinations (of which there are 10). To equate the total number of input units with the first version, 3 redundant bits were used for each of the 5 bits, for a total of 15 (see Figure 6.1b).

Figure 6.2 shows a depiction of the architecture, which simply has the four input and output slots, with a hidden layer of 49 units, and the LEABRA networks had the standard 25% activity constraint. Standard parameters as in Chapter 5 were used for training the networks. As explained in the taxonomy of tasks, it is expected that associative learning will *not* be particularly beneficial for generalization on this task, as it will tend to bias the representations to pick up on spurious correlations across slots. The basic LEABRA results are for an associative learning strength of .1, and results without associative learning at all (No As or ReBel + CHL) are also shown.

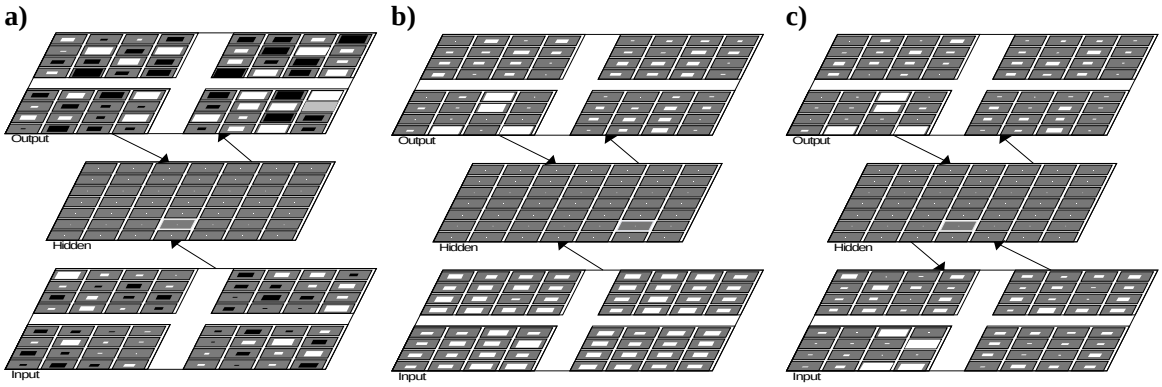

Figure 6.2: Hidden unit weight patterns in version 1 of the combinatorial domain task (relatively non-overlapping units), with both input-to-hidden and output-to-hidden weights are indicated in the corresponding input or output unit, with the area indicating magnitude and color indicating sign (white = positive, black = negative). **a)** Shows the weights for a typical CHL hidden unit (BP units looked similar, not shown). Note that there are significant weights in each of the four different input feature slots. **b)** Shows weights for a typical LEABRA hidden unit, showing clear slot-based specialization for the output-to-hidden weights, but only a weak indication of this for the input-to-hidden weights. **c)** Shows weights for AE LEABRA, which, by virtue of having error-driven learning in representing the input, has clearer slot-based input-to-hidden weights than the standard LEABRA network, but similar output-to-hidden weights.

Version 1: Relatively Non-overlapping Patterns

To assess generalization performance in these networks, either 100 (small set) or 500 (large set) randomly generated combinations of values over the four slots were generated for the training set, and 500 different such combinations were generated for the testing set. Networks were run for 400 epochs, and generalization measured every 25. The types of hidden unit weight patterns that developed in this task (small training set) are shown in Figure 6.2. The CHL network exhibits essentially the same kind of noisy, under-constrained representations as in the digits task, while the LEABRA networks have more clearly represented both the independence of the different input slots, and the mapping function between input and output patterns. Note that both the visual clarity and generalization efficacy of the input-to-hidden weights in LEABRA appears to improve in the AE (auto-encoder) version, due to the error-driven learning pressure on those weights in this condition.

Figure 6.3 shows the generalization performance for the basic algorithms on the large training set (500 patterns), which replicates, rather dramatically, the basic pattern seen in the digit recognition task. Thus, as before, the BP network is capable of quite good generalization despite not having units that appear to represent the basic independence of the different slots, while the LEABRA network also achieves good generalization by virtue of having units that clearly represent the important structure of the task. In contrast, the CHL networks, which have similar weight patterns to the BP networks, have dramatically worse generalization performance, presumably due to the negative interaction of interactive processing and underconstrained weights.

The figure shows results for a number of different parameterizations of CHL to attempt to produce better generalization. A faster learning rate (.1) did improve generalization over the slower

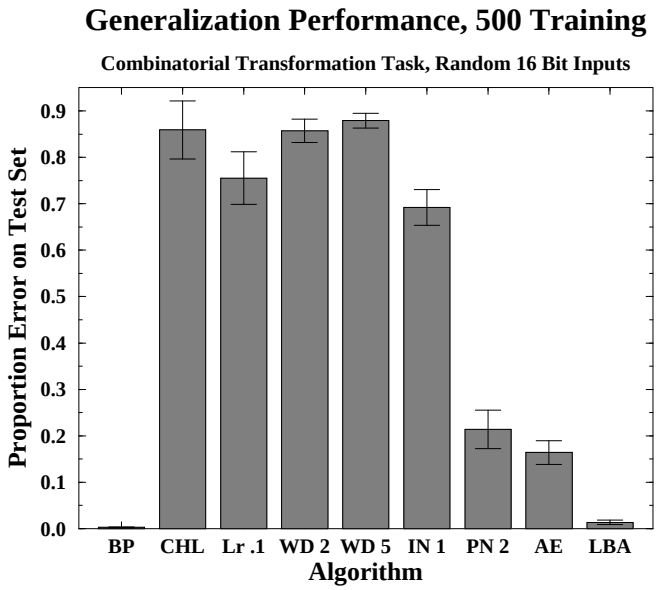

Figure 6.3: Generalization performance of the different algorithms on the large training set (500). BP and LEABRA (**LBA**) generalization are essentially perfect, while CHL generalization is quite poor, both relatively and absolutely. The use of the faster learning rate (**Lr .1**) appeared to yield some improvement relative to the standard slower learning rate. All subsequent CHL results are with Lr .1. The use of weight decay (**WD**) at the 2 (.002) and 5 (.005) levels only impaired performance. Some improvement was observed with .1 input noise (**IN 1**). Substantial, but still not to the level of BP or LEABRA, improvements were obtained with .2 variance processing noise (**PN 2**) the **AE** auto-encoder version of CHL.

one, this effect was not substantial. Weight elimination actually impaired performance on this task, which is discussed further below. The use of noise in the inputs (*IN*) resulted in some improvement at the .1 level, but .2 variance noise prevented any network from learning to criterion. The use of processing noise, in contrast, produced significantly better generalization at the .2 level, despite the fact that this prevented three out of five networks from learning to criterion (though they came very close). Finally, the auto-encoder (AE) method of training CHL led to even greater improvements in generalization, but even it was still significantly worse than BP and LEABRA. This improvement for AE CHL in this task contrasts with the results on the digit task, where AE performed worse than standard CHL. In that task, AE training was thought to result in overfitting the noisy details of the digit images, which detracted from any benefits it might have otherwise provided. There is no sense in which the present task has noisy input patterns, and thus this problem of overfitting does not arise, allowing the benefits of the AE procedure to be seen.

The reasons for the generally poor generalization performance of CHL will be investigated in detail below. Before doing so, a more detailed comparison between BP and LEABRA can be made. The results of BP and LEABRA on the large training set version of this task were sufficiently good that it is necessary to use the smaller training set (100 patterns) in order to measure meaningful differences in generalization performance. Results for this training set are shown in Figure 6.4. The relationship between the three standard algorithms (BP, CHL, and LEABRA) is basically as before, with

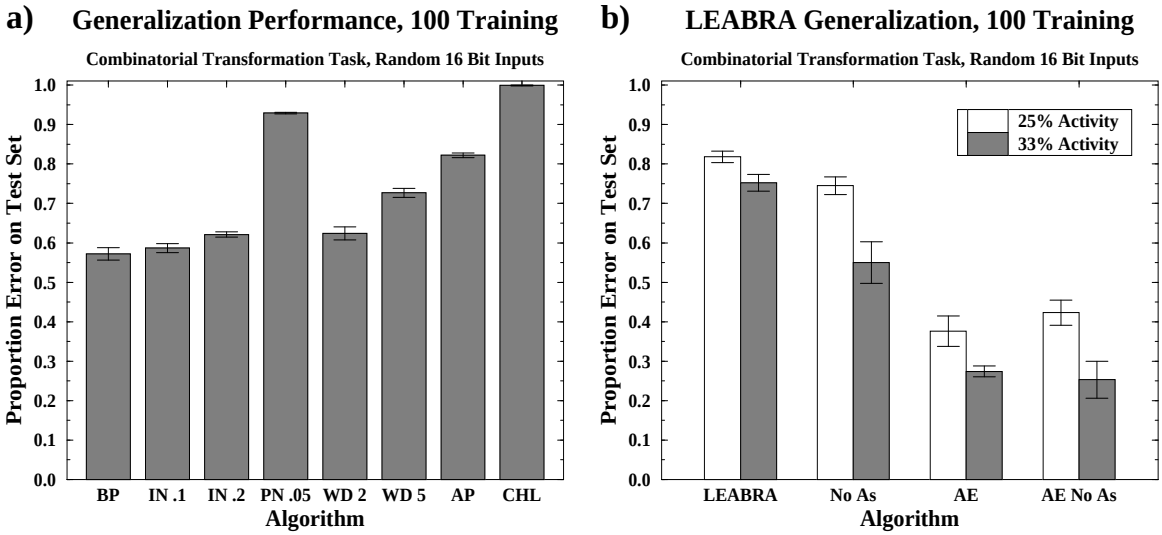

Figure 6.4: Generalization performance of the different algorithms on the small training set (100). **BP** is standard backpropagation. **IN** is input noise, and **PN** is processing noise, with variances as indicated. **WD 2** is BP with .002 weight-elimination, and **WD 5** is with .005. **AP** is Almeida-Pineda, which was intermediate between BP and **CHL**, which essentially did not generalize at all. **LEABRA** is standard LEABRA, and **No As** is LEABRA without associative learning (ReBel + CHL), which performs better as expected. **AE** is auto-encoder LEABRA, and **AE No As** is AE LEABRA with no associative learning. Two activity levels are shown, the standard 25%, and a higher activity level of 33%, which performed better. Note that associative learning was not a liability for AE LEABRA.

CHL exhibiting a floor effect (essentially no generalization at all). However, it is clear that feedforward BP generalizes better than standard LEABRA in this task. A number of interesting conclusions can be drawn from this and the other data presented in the figure.

MaxIn associative learning is not useful in this task: As was expected, the use of associative learning actually impaired generalization performance in some cases in this task. The negative impact of associative learning in this task probably arises because of its tendency to encode correlations, which are spurious in this case, between different input slots. Thus, given the small training sample used, it is the case that certain patterns co-occur in the same slots relatively frequently. Associative learning will tend to form representations that link these patterns, which, given that the correlation does not hold in the underlying domain, will result in poorer generalization performance. However, it is interesting to note that in the AE condition, associative learning appeared to reduce the variance and improved generalization slightly in the 25% activity case. Further, when 500 training patterns were used, associative learning appears to provide a small generalization advantage (with associative learning is .0136, SEM .00455 compared to .0272, SEM .0116 without).

Standard LEABRA suffers from having sparser, conjunctive representations: The reason for the disparity between BP and LEABRA on this task is due in part to the ReBel activity constraints in LEABRA, which lead to sparser and potentially more conjunctive representations. To the extent that individual units tend to represent the conjunction of patterns across multiple slots, this will impair generalization performance since the slots are actually independent and should be represented

Algorithm	Initial Weights	Trained Weights	Cycles	Gen Err	SEM
BP	—	—	—	.572	.0155
AP	.252	.274	58	.822	.0061
CHL	.252	.704	140	.999	.0011

Table 6.1: Comparison between measures of interactivity and generalization in AP and CHL. Weights are average absolute value of reciprocal (top-down) weights from output to hidden units in Almeida-Pineda (AP) and CHL networks. The CHL network clearly increases the strength of these weights over training, while the AP network does not. CHL networks take significantly longer to settle during processing, as is shown in the Cycles (max) column (initial settling time before training was 48 cycles). The level of interactivity between BP (none), AP, and CHL is consistent with their respective generalization error scores.

independently. Thus, increasing the activity constraint from the standard 25% to 33% resulted in significantly better performance in this task (note that 25% was optimal for the digits task). Thus, it is clear that the activity level parameter is somewhat task-dependent.

Weight decay is not effective in this task: The effects of weight decay in both the feedforward BP and CHL networks were consistently negative for all values of decay strength tested. Further, the size of the negative impact appeared to be correlated with the magnitude of the decay strength. This result is inconsistent with those of Brousse (1993), and may indicate a dependence on the nature of the input-output patterns for the advantages of weight decay. While Brousse (1993) used a simple auto-encoder, where the input and output patterns were the same for each slot, in this task a mapping function between input and output patterns had to be learned. It seems likely that weight decay interferes with the learning of this mapping task, but not the simpler auto-encoder task.

Noise is not effective for BP in this task: Noise either in the inputs or in processing (added to all units) did not improve generalization performance at all for BP in this task. In fact, as before, adding noise in the processing significantly impaired generalization, and a variance of .1 prevented BP from learning the task to criterion. As was the case in the digits task, the lack of effectiveness of noise is probably due to the binary nature of the input and output patterns in this task. However, this conflicts with the results from the CHL network with processing noise. Perhaps the difference lies in the number of training patterns used (100 *vs* 500), but this is difficult to test given that BP generalizes near-perfectly with 500 patterns without any additional manipulations.

Almeida-Pineda is intermediate between BP and CHL: As was the case in the digits task, the generalization performance of the AP algorithm is consistent with the idea that it is less interactive than CHL, but (obviously) more interactive than BP, and that interactivity is the source of the generalization problems with CHL. Table 6.1 shows data for this task that, as before, supports this interpretation. The magnitude of the reciprocal (output-to-hidden) weights after training in the AP network is only slightly different than before training, while those in the CHL network have increased considerably. The number of cycles taken to settle is also indicative of the increased interactivity of the CHL network.

Auto-associator LEABRA performed significantly better than other algorithms: As was the case

on the digits task, the use of error signals to shape the development of representations of the self-structure of the input and output patterns resulted in substantial gains in generalization performance. As is evident in Figure 6.2c, the weights for AE LEABRA provide cleaner representations of the independence among the slots in the input layer, which is clearly reflected in its generalization performance relative to standard LEABRA. This figure emphasizes the point that the advantages of AE come in applying error-driven learning pressure to the same weights which are used in the input-output mapping task.

Self-organizing learning only (No Err) cannot solve this task: An attempt to train a LEABRA network without the CHL error-driven learning component failed. There was clearly evidence of learning in terms of the sum-squared-error (SSE), since the initial SSE was around 1800, and the final was around 700. However, in terms of the number of patterns with all output units on the correct side of .5, the network showed little improvement over learning, remaining at around 100 for the 100 pattern version of the problem. This confirms that error-driven learning is essential for solving even fairly simple tasks such as this one, and emphasizes the importance of error-driven learning for understanding how the neocortex can learn complex tasks. Finally, it is not surprising that generalization was nonexistent in this network, given that it had not learned the task in the first place.

Why CHL Generalizes So Poorly

In order to test the theory that heightened sensitivity to differences in input patterns (due to interactive processing) is behind the poor generalization performance of CHL, an attempt was made to directly measure this sensitivity. This test involved presenting pairs of input patterns from the combinatorial domain which differed in only one of the four input feature slots, and recording the correlations between the two hidden unit activity states corresponding to these pairs of inputs. If the network is exhibiting a graded, proportional response to the 75% similarity of the input patterns, it should have a correlation of approximately .75 across the two hidden unit patterns for these two input patterns. A lower correlation would indicate a heightened sensitivity to the differences in the input patterns (i.e., the butterfly effect), while a higher correlation would indicate a decreased sensitivity (i.e., attractor dynamics where both patterns fall into the same attractor).

The results of this test are shown in Figure 6.5, which confirms the hypothesis that the CHL networks suffer from heightened sensitivity to input differences. The first column in the figure shows the hidden unit correlations across pairs drawn from the training set (i.e., the networks had already been trained on these patterns). In this case, all three algorithms have hidden unit correlations which are essentially proportional to the difference in the input, as would be expected since they have been trained to produce the correct outputs for these patterns, which also differ by 75%.[1] However, the second column, which shows pairs of patterns in the testing set (i.e., novel patterns), indicates that while BP and LEABRA remain in the proportional response range (near .75), the CHL network hid-

[1] It is interesting that the LEABRA case shows some indication of a common attractor for the two patterns, which might indicate the existence of higher-order representations that respond to combinations across different input patterns.

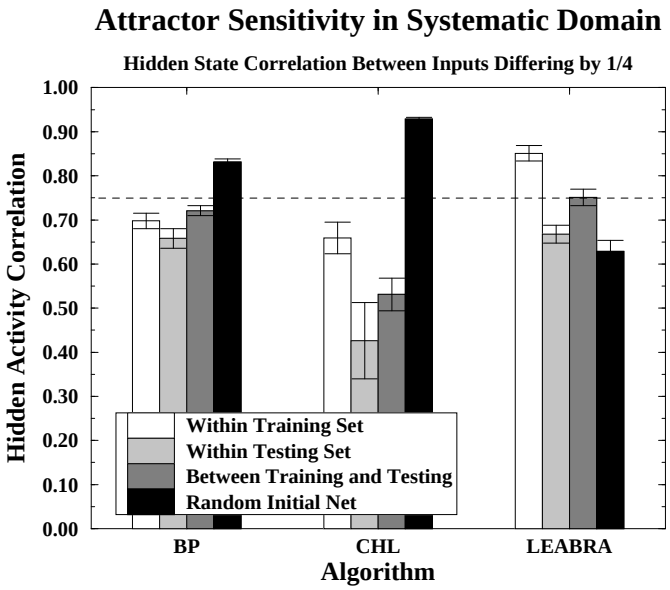

Figure 6.5: Response of the different networks to two different input patterns differing in one out of the four feature slots, measured as the correlation across the corresponding hidden unit patterns for the two patterns. If the hidden units are representing the inputs proportionally to their differences, they should have a correlation of .75 as shown by the line in the figure. Results are averages over 9, 12, 24, and 45 different patterns, for the conditions in the order shown.

den unit correlations drop dramatically to around .42. Thus, the network is responding quite differently to each member of these similar, but novel, input pattern pairs. Given that the trained items did not display this sensitivity, it seems likely that the culprit is the under-constrained nature of error-driven learning, which did what was necessary to get the trained items be processed correctly, but the residual noise in the weights, combined with the interactive activation updating, leaves the network highly sensitive to differences in novel items.

The third column in the figure tests the idea that when presented with one familiar and one novel pattern which differ by only one feature slot, the CHL network will settle into a common attractor state for both patterns (presumably corresponding to that of the trained pattern). However, the results indicate that, while the states are not as different as between two completely novel inputs, they nevertheless reflect settling into distinct attractor states. BP and LEABRA remain essentially proportional in this case. Finally, the fourth column shows the pairwise differences over all patterns for an untrained (random) weight configuration. In this case, the CHL network does settle into a common attractor for the two patterns, while the BP network has some tendency to exhibit the same hidden unit state, and LEABRA has the most differentiated hidden unit patterns of the three. This increased initial differentiation in LEABRA probably results from the activity competition imposed by the ReBel kWTA activation function.

In summary, there is clear evidence for greater sensitivity in CHL to differences in input patterns that prevent it from exhibiting a graded, proportional response to novel inputs. This is in contrast to the LEABRA network, which is equally interactive. It appears that the soft kWTA constraint im-

Algorithm	$< .05$	$> .95$	$< .1$	$> .9$
BP	7.6	8.02	10.58	11.45
CHL	15.25	14.54	16.63	15.66

Table 6.2: Average number of hidden units (N=40) above or below given activation value on the training items in a fully trained network. The hidden units in the CHL networks clearly have more extremal values, which could have contributed to poor generalization.

posed by the ReBel activation function damps the interactive activation dynamics, resulting in a kind of proportionality in the activation patterns that is more similar to that of the feedforward BP network than the CHL network. This is undoubtedly an important contributing factor for why LEABRA generalizes as well as it does. The greater sensitivity of the response of the CHL network is consistent with the hypothesized explanation for its poor generalization performance. However, it is possible that other factors such as the tendency for the units in the CHL network to be in the non-linear range of their activation function also play a role.

To test this latter hypothesis, the average number of hidden units with activation states in their non-linear range was computed over all the training patterns on fully trained networks. The results are shown in table 6.2. Thus, the CHL networks have more units with extremal activation values, which could be contributing to the lack of a graded, proportional response to novel stimuli. However, it is likely that both of these phenomena, greater sensitivity induced by non-linear attractor dynamics and extremal activation states, are two sides of the same coin.

Failure of Adapting Activity Regulation

Since the associative learning in LEABRA was not particularly useful, it is the activity constraints provided by the ReBel kWTA function that was largely responsible for good generalization performance on this task. Thus, it is of interest to determine if the adapting activity regulation mechanisms described in Chapter 5 for CHL will lead to an improvement in its otherwise terrible generalization performance. However, it turns out that the adapting mechanisms did not work at all on this task — when the activity constraint was made strong enough to potentially affect the activity levels, the networks became unstable. Further, the constraint was ineffective even at the strong level. Thus, there is apparently no value of the activity constraint weighting parameter for which the activity constraint was effective and the network stable.

The problems with the adapting activity constraint are illustrated in Figure 6.6, which shows the activity level and training error for two values of the activity constraint weighting factor k_{actlim} (1 and 1.5). For both cases, there is an initial period of training where the network solves the problem, but the activity levels are not close to the .25 target value, and generalization is no better than in a standard CHL network (not shown). Then, as training proceeds, both activity levels and error scores become wildly unstable. This happens earlier for a larger value of the activity constraint. It is likely that the accumulated pressure of the activity constraint, which is apparently not compatible with the error-driven pressure to solve the problem, leads to the instability. Comparing the weights

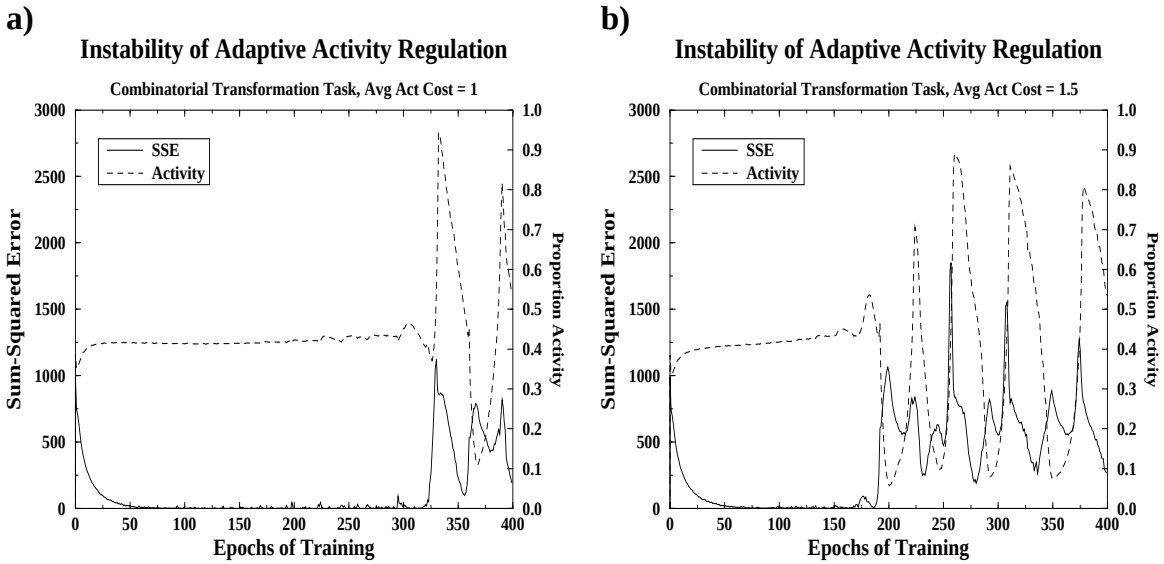

Figure 6.6: Training error and activity levels for activity regulation in CHL networks, with a target activity level of .25. a) shows the case with an average activity constraint with strength 1, which initially learns the problem successfully, but has too high an activity level, does not show improved generalization, and eventually suffers from instability. b) shows the case with strength of 1.5, which is similar to a) but the instability occurs earlier in training. Larger values result in even earlier instability onset, while smaller values result in no appreciable impact on activity level.

in an activity regulated network to those in a standard CHL network, the main difference appears to be a larger magnitude of bias weights, but the other weights appear roughly similar. Thus, it is not the case that the instability results from the weights being either extremely large or near zero.

This failure of the adapting activity regulation method lends further support to the implementational principle favoring fixed constraints, such as the ReBel kWTA activation function used in LEABRA. Given the overall lack of promising results from the adapting method in this and the digit recognition tasks, no further exploration of this method will be performed on subsequent tasks.

Other Parameters Affecting Generalization Performance

There are several parameters of LEABRA and the other algorithms that have an effect on the generalization performance. For BP, the number of hidden units had a significant impact (40 units: .744 generalization error, SEM .0117, 49 units (reported above): .572, SEM .0155, 64 units: .462, SEM .0112). This effect was not replicated in the CHL networks, which had uniformly negligible generalization performance on the small training set used. As reported above, weight elimination weight decay did not have a favorable impact on the generalization performance of BP, and no significant impact on CHL. Finally, using a learning rate of .1 instead of the standard .01 resulted in slightly worse generalization in BP (.627, SEM .00932).

This task did show a sensitivity to the use of the GausSig *vs* sigmoidal likelihood function in LEABRA. The sigmoidal networks did not generalize nearly as well, with a generalization error of

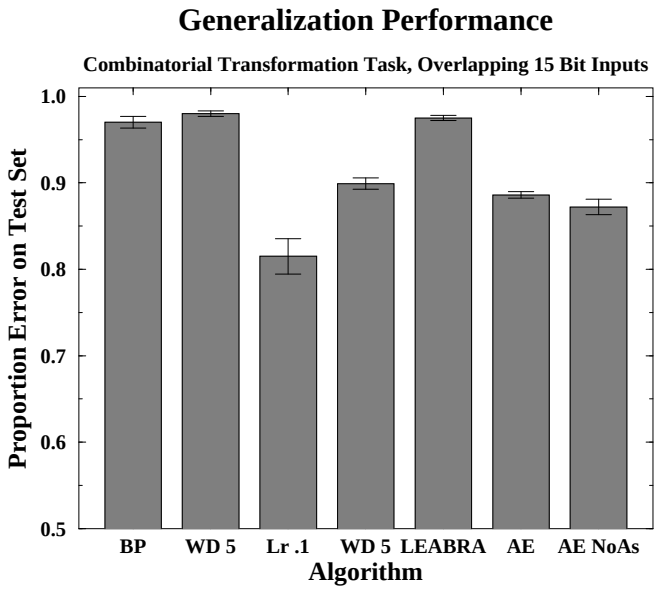

Figure 6.7: Generalization performance of the different algorithms for overlapping patterns. **WD 5** is preceding column plus weight-elimination weight decay at .005. **Lr .1** is learning rate of .1. **AE** is auto-encoder LEABRA, and **AE No As** is AE LEABRA without associative learning (ReBel + CHL).

.741, SEM .0123, in the auto-encoder (AE), 33% activity, no associative learning case, compared to .253, SEM .0467 for GausSig, which is what is reported above. This is probably because of the greater specificity of the response for GausSig units, which means that they are more likely to be affected by weights which mismatch the inputs they respond to, and thus more likely to develop weights specific to a given independent feature.

Version 2: Overlapping Patterns

This second version of the combinatorial task was used in order to test the performance of the algorithms with a more difficult mapping task between input and output patterns within a given slot — one that unambiguously requires non-linear transformations. This might cause the BP networks to develop more non-linear representations which would then not show the graded, proportional response characteristic of the previous tasks, resulting in worse generalization performance. In contrast, the LEABRA networks might be more immune to the increased difficulty given that they already tend to represent the inputs in terms of the mapping between entire input/output feature patterns.

Figure 6.7, which shows generalization performance for BP, LEABRA, and LEABRA AE algorithms on this version of the task, indicates that this hypotheses might be at least partially correct. For the BP networks that use the same learning rate parameter as in the previous version (.01), generalization is indeed quite poor, both with and without weight elimination. However, when a faster learning rate is used (.1), generalization improves significantly. The LEABRA network performed at the same level as BP, which was not the case on the easier version of this task. The auto-encoder

form of LEABRA performed better, as before, but still not to the level of the .1 learning rate BP network. The use of associative learning in AE LEABRA appeared to be somewhat detrimental given the *No As* results. However, in the feedforward case, it actually improved performance, since the No As version was not even capable of learning the task to criterion (not shown). Unlike the previous task, the use of a higher activity level (33% instead of the standard 25%) did not improve generalization performance. Finally, the use of a faster learning rate in LEABRA (.02 instead of .01) did appear to improve generalization (.953, SEM .00551 compared to .975, SEM .00297) but 2 out of 5 networks did not learn to criterion.

One hypothesis regarding the effect of learning rate in BP is that the larger learning rate allows hidden units to develop more differentiated representations, that are more likely to represent the individual slots, whereas the slower learning rate results in more overlapping representations that represent multiple slots. However, a visual inspection of the weight patterns did not reveal any obvious difference along these lines. Thus, the reason for the apparent advantage of the faster learning rate, which was not found on any of the other tasks studied thus far, remains somewhat unclear.

Note that weight decay for BP networks was not advantageous on this version of the task either, probably for a similar reason as before: it interferes with the learning of the individual slot mapping task. Finally, it should be noted that none of these networks is performing anywhere near the level that they did in the easier version of the task, indicating that the difficulty of the mapping task is an important variable in determining overall generalization performance. Another variable which has been observed in work not reported in this thesis to affect performance in LEABRA is the overall activity level in the input and output layers. If this significantly exceeds the nominal 25% level, LEABRA's learning and generalization performance is generally impaired. The 40% activity level of the patterns in this task is thus a potential problem for LEABRA.

Discussion

The results on the above versions of the combinatorial transformation task are generally consistent with the previous results on the digit recognition task, and with the overall set of hypotheses regarding the nature of learning and generalization in the LEABRA algorithm as compared to standard error-driven algorithms. With respect to the central functional criteria for a neocortical learning algorithm identified in the introduction, these results support the contention that standard algorithms fail to exhibit both interactive activation dynamics and effective use of distributed representations for systematic behavior on novel stimuli. That LEABRA is capable of exhibiting both of these properties supports the idea that it should be considered a more plausible candidate for describing neural learning and processing in the neocortex. The next section provides additional evidence that is consistent with the previous findings. The discussion is then extended to consider the generalization of knowledge *across* tasks, not just within a given task.

Combinatorial and Relational: The Lines Task

The horizontal and vertical line task, studied in various forms by a number of researchers (Földiák, 1990; Saund, 1995; Zemel, 1993; Dayan & Zemel, 1995), involves a simple "retinal" input representation which can contain some number of horizontal and vertical lines. One form of this problem uses an auto-encoder task, where the objective is to reproduce the input pattern over the output layer, while another simply requires the activation of output units which identify the orientation and position of the lines. Both are transformational tasks according to the taxonomy presented in the previous chapter. The latter form of the task is used here, to enable the additional use of the auto-encoder form of the LEABRA network to be evaluated without redundancy in the task. Since each line is made up of input unit activities that are also used in other lines, the identification of a given line requires relational information. However, the identification of a given line is relatively independent of the identification of other lines, making it also a combinatorial input. This mix of the two input types is part of what makes this an interesting problem.

It is expected that, unlike in the combinatorial domain studied above, the associative learning in LEABRA will be important for obtaining good generalization on this task, since it will facilitate the development of representations which capture the correlational structure that defines an individual line. Finally, it should be noted that several other algorithms have been tried on this problem, either with activity constraints (Saund, 1995; Dayan & Zemel, 1995) or a self-organizing learning algorithm (Földiák, 1990), so that LEABRA's performance can be compared to that of these other algorithms.

While a number of different versions of this task have been studied, in the version we consider here the environment consists of all possible combinations of two horizontal and/or vertical lines, each of which is 5 pixels long and placed within a 5x5 grid. Thus, for this size grid, there are $\begin{pmatrix} 10 \\ 2 \end{pmatrix} = 45$ unique combinations of two lines. There are 10 output units, one for each possible line position and orientation. Networks are trained to activate the units corresponding to the lines present in the input. For the following generalization results, 10 out of the 45 total possible line combination patterns were selected at random for the testing set, leaving the remaining 35 for the training set. 5 different such training/testing sets were created, and the one with the best generalization performance in a standard BP network used for evaluating the other algorithms (as it happens, there were largely insignificant differences between the generalization and other performance measures over all 5 sets).

Figure 6.8 shows that, as in the previous tasks, the representations formed by LEABRA capture the underlying regularity of the lines domain, namely, lines. In contrast, the CHL network again shows the same kinds of underconstrained, noisy representations as were seen on previous tasks (BP had similar representations). It is interesting to note that even though the activity constraint in LEABRA was set to 6 units in this case (25% activity), only 4 were typically active. There were

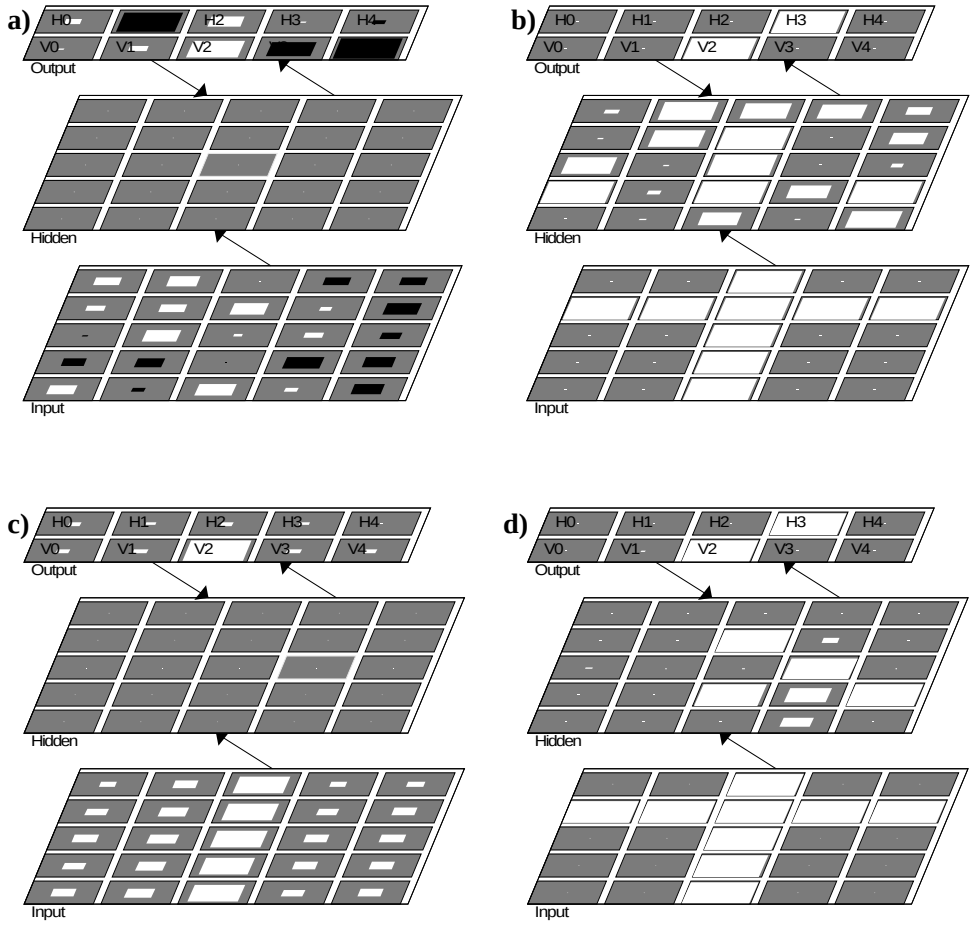

Figure 6.8: Comparison between the weights and activation patterns for CHL and LEABRA networks. **a)** shows the weights for a typical CHL hidden unit, which appear quite noisy, but the output weights indicate that it encodes the V2 (vertical, position 2) line. **b)** shows the activations when a pattern containing V2 is presented, confirming that this unit participates in the representation. **c)** shows the weights for a typical LEABRA hidden unit, which clearly encodes the V2 line. **d)** shows the activations for the same input pattern as in the CHL network. Despite the fact that the activity level is set to 6 active units, the LEABRA network usually has only 4 active, with a two-hidden unit redundancy for each of the ten lines (and thus 5 units which do not actively participate in any representations).

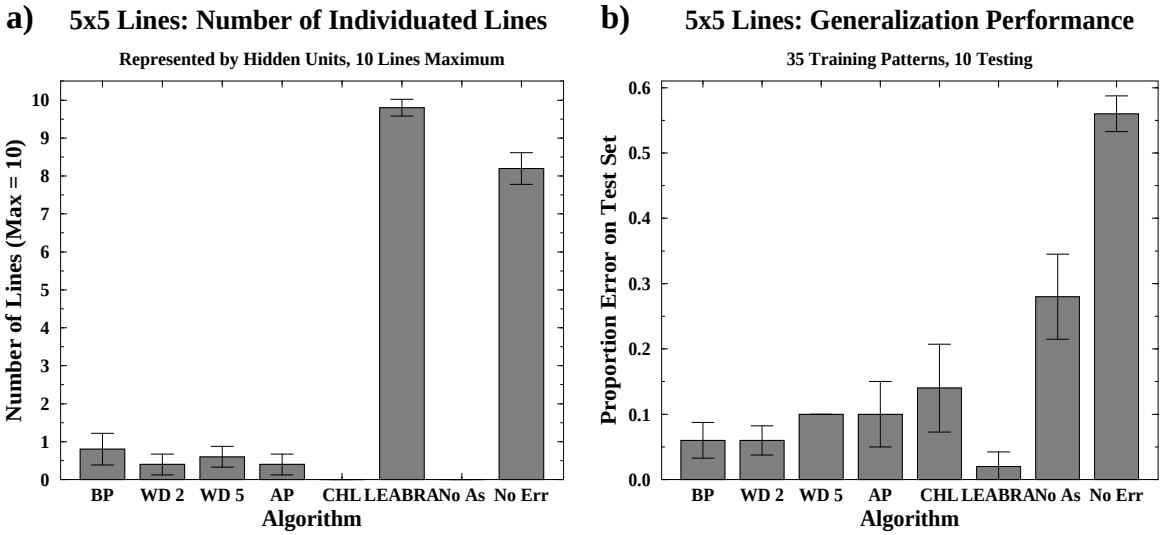

Figure 6.9: **a)** shows number of individuated lines (method described in text) for the different algorithms. The associative learning in LEABRA is obviously critical for developing individuated line representations. **b)** shows the generalization performance of the different algorithms, which is largely consistent with results on other tasks. Again, weight elimination (**WD 2** for .002 and **WD 5** for .005) was not effective. Associative and error-driven learning together result in better generalization than either alone (**No As**, **No Err**). Also, note that the **No Err** networks (ReBel+MaxIn) took substantially longer to learn the task to criterion than LEABRA with error driven learning (145 epochs *vs* 8).

2 units coding for each of the 10 different lines, so each pattern of 2 lines resulted in 4 active units. Thus, this is an example of the individual likelihood term in the ReBel activation function causing the activity level to be below the specified upper limit, and illustrates the advantages of this feature (see Chapter 4 for discussion).

In order to quantify the extent to which the hidden layer units represent individual lines, the correlation between each hidden unit's weight vector and each of the individual horizontal and vertical lines was measured. After computing these correlations, the highest correlation value over all units for a given line was computed, and a thresholding was applied to these values. A maximal correlation that was greater than .8 was counted, indicating that the line in question was represented individually by one of the hidden units. Thus, the maximum count is 10, one for each line. The results for both this line count measure and the generalization error are presented in Figure 6.9.

This figure shows that the associative learning in LEABRA is critical for the development of individuated line representations in the hidden units, since only in the LEABRA (with .2 associative learning) and LEABRA No Err cases are there significant numbers of such representations. Note that the error-driven component is important too, since the LEABRA networks had more individuated line representations than the case without error-driven learning (No Err). The number of individuated lines is not, however, a completely reliable predictor of generalization performance, since the LEABRA No Err case had the worst generalization of any algorithm, while standard LEABRA had the best. These results again show that it is the interaction between error-driven and associative

N Units Active	Gen Err	SEM	N Indiv Lines	SEM
2	0	0	10	0
4	.04	.0274	10	0
6	.02	.0224	9.8	.224

Table 6.3: Effect of activity level on generalization performance and number of individuated lines represented by the hidden units in LEABRA. The 6 active units (25%) case is what was reported above.

learning in LEABRA that gives it its good generalization performance, and helps to produce better hidden unit representations. This is consistent with the results on the digit recognition task, which shares with this task the relational quality of the input patterns, which is where LEABRA is expected to give the greatest benefits.

As was the case in previous tasks, the generalization performance of BP does not depend on the presence of representations that capture the basic structure of the task (lines in this case). Again, weight elimination did not have a significant impact on performance, either in generalization or on the number of individuated lines represented. As has been the case with the previous tasks, the CHL network generalizes worse than the BP network, despite having comparable representations. This is probably due to its butterfly-effect sensitivity, as discussed in the previous section. Also consistent with previous results, the Almeida-Pineda networks (AP) performed at a level intermediate between that of BP and CHL.

For the BP networks, using a faster learning rate led to better generalization performance (.02, SEM .0224) for learning rate of .1, compared to .06, SEM .0274 for a learning rate of .01. However, the effect was the opposite for the CHL networks (.01 lrate: .14, SEM .0671, .1 lrate: .22, SEM .0418). Also, using only 10 hidden units in BP, which might in theory force more individuated representations, did not have this effect. Generalization error was worse at .3, SEM .0791 in this case, with .8, SEM .418 individuated lines represented, compared to the .06, SEM .0274 generalization and .8, SEM .418 individuated lines in the case with 25 hidden units (shown in figure).

In theory, only two active hidden units are necessary for solving this task. As was observed above, LEABRA will reduce the number of active units it uses from 6 (25%) to 4 as the hidden unit representations develop over learning. If the activity level is set from the start to a reduced level, performance is roughly comparable, but there is an apparent improvement in the number of individuated lines represented (see Table 6.3 for results). Since the standard LEABRA network performs so well on this task, there is little room for improvement for the AE version, which nonetheless performed perfectly on this task (0 generalization errors, all 10 individuated lines).

Finally, these results can be compared with those obtained by other researchers on versions of the lines task. In the algorithm explored by Dayan and Zemel (1995), which enforced a form of activation competition, they found a failure rate of 39 out of 100 networks for the development of individual line representations, and a related algorithm studied by Saund (1995) had a failure rate of 75 out of 100. This further confirms the idea that the associative learning in LEABRA, which is not

present in either of these activation-competition based networks, is important for this task.

Cross-Task Generalization

The results presented so far support the hypothesis that the ability to treat novel items systematically (i.e., generalization) can be improved by developing representations that capture the structure of the environment. In the standard LEABRA algorithm, the associative learning causes the hidden units to represent the correlational self-structure of the input and output patterns, as well as the input-output correlations. These representations lead to better generalization performance by virtue of being aligned with the structure of items in the environment, as compared to the under-constrained representations that develop in purely error-driven algorithms. Further, the introduction of error signals into the learning of the self-structure of the environment through the AE LEABRA algorithm resulted in even better generalization performance.

This same argument can be extended, perhaps even more significantly, to the case of generalization across tasks. To the extent that the representations developed in learning one task capture the general structure of the environment, a novel task operating in this same environment should be more easily learned, compared to the case where the representations are merely sufficiently discriminative for solving the original task, but do not otherwise capture the structure of that environment more than is necessary to solve that task. To make this more concrete, take the case of the BP network on the lines task described above. While it is clear that the BP network, as an aggregate system, exhibited behavior consistent with the regularities of the lines environment, the individual representational elements within the system did not reflect these regularities. Thus, if the system were to learn a novel task which recombined the basic features of the environment, the lines, in novel ways, the confounding of multiple line representations in the BP network will lead to interference and inability to learn. On the other hand, given that LEABRA has extracted the line elements successfully, it should be able to use its representations to learn any given arbitrary recombination of the lines.

To test this idea, the hidden unit activities for the LEABRA, BP, and CHL networks were used as inputs to a second network which was trained to turn on one output unit if any of the lines in the input was in an odd position (1,3,5), and to turn on the other output unit otherwise. This is a simple classification task, and it can be learned easily in a single layer network, which is what was used. Because the BP (and CHL) representations confounded several lines in a given representation, this made the classification task more difficult, and these networks took nearly three times as long to train (see Figure 6.10). The CHL representations tended to be even less systematic than the BP ones, resulting in slower and also highly variable training times.

This test was replicated in the digit recognition task, by training LEABRA and BP networks on the original categorization task, and simultaneously training on an additional even-odd categorization task, where one unit was active if the digit was an even number, and another was active if it was an odd number. This is perhaps a more “naturalistic” test of cross-task generalization, as it occurs

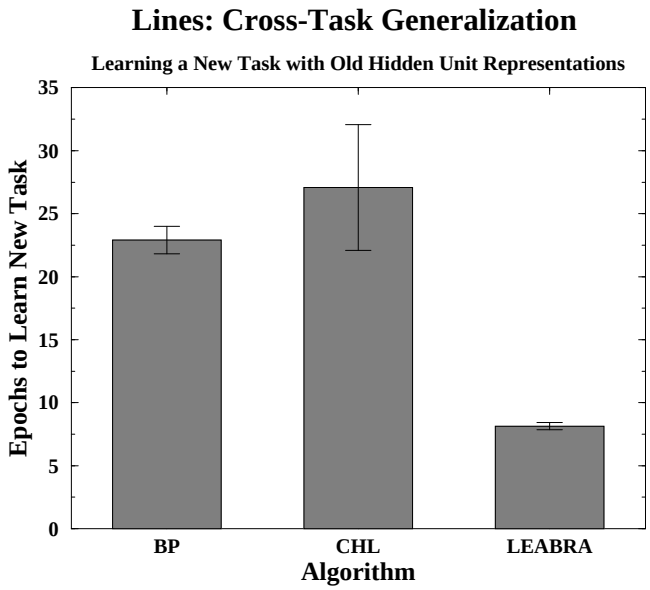

Figure 6.10: Cross-task generalization of the hidden unit representations to the odd-even line categorization task. Hidden unit activities from the standard 5x5 line task were used as input to a two-layer BP network which had to categorize the lines as odd or even. The mean training time required (in epochs) is shown (computed over 10 runs each of 10 different hidden unit patterns). The systematicity of the LEABRA patterns enables this second task to be learned nearly three times as fast as BP. The CHL patterns resulted in highly variable training times.

on-line and in the same network, learning a relevant property of digits. In order to test the ability of representations learned in the original digit categorization task to transfer to the even-odd task, without these representations themselves being influenced by the learning of that task, no error signals were propagated from the even-odd categorization output layer back to the hidden layer. This can be done in LEABRA simply by not including a return projection from that output layer to the hidden layer (using only the feed-forward one). In BP, the connections were explicitly made to not contribute to the derivative of the error with respect to the hidden unit activity state. Thus, the even-odd task had to be learned using only the weights from the hidden layer to the even-odd output layer.

Figure 6.11 shows the results of this test, indicating both learning time on the original digit classification task, and on the even-odd task, to a criterion of all units on the right side of .5 for all training patterns. LEABRA learned the easier even-odd task before the digit categorization task (though the error is very low at the point when it gets the even-odd task right, it can take a while to get to perfect performance). BP failed to learn the even-odd task to criterion in 3 out of 5 networks. Given that it had thoroughly learned the digit classification task by the 200 epoch training cutoff point, it is unlikely that these 3 networks would have ever learned the even-odd task. However, for graphing purposes, they were scored as 200 epochs (the two that did learn were at 188 and 109 epochs). Thus, it appears that the representation of the self-structure of an environment in LEABRA can significantly facilitate the ability to learn novel tasks in that environment.

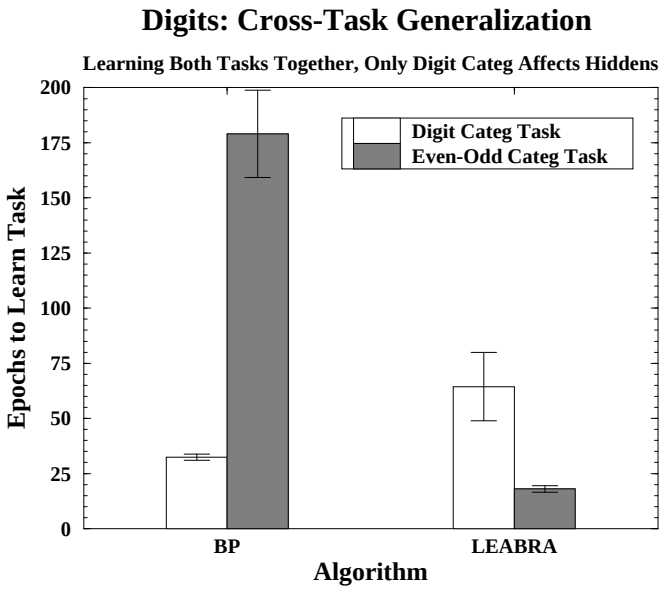

Figure 6.11: Cross-task generalization of the hidden unit representations to the odd-even digit categorization task. Both tasks were learned simultaneously, but only the standard digit categorization task affected the hidden unit representations. Thus, one layer of adapting weights was available to learn the even-odd task. LEABRA learns this easily (it is an easier task than the digit categorization), but BP does not. Indeed, 3 out of 5 BP networks had failed to learn the even-odd task after training was stopped after 200 epochs (these were scored as 200, though they probably would never have learned).

Discussion

In combination with the results on the digit recognition task studied in the previous chapter, the results presented above provide a largely consistent body of evidence in support of the major hypotheses regarding the characteristics of the LEABRA algorithm in comparison to standard algorithms. In addition to the results on generalization performance, which is facilitated by the associative learning and activity constraints in LEABRA, the finding that the representations developed by LEABRA provide a better basis for learning novel tasks is potentially important for models of human cognition. The ability to use knowledge flexibly has obvious adaptivity for animals, and is clearly an important characteristic of human cognition.

Chapter 7

Learning in Deep Networks

One of the most important developments in neural networks was the ability to solve problems using hidden units with the error backpropagation algorithm (Rumelhart et al., 1986a). This was important because, in many cases, difficult problems become easier to solve when multiple stages of representations are used. Also, it is clear that the brain uses multiple stages of representations in solving difficult problems like visual object recognition — the neurons in the inferior temporal cortex that appear to code for objects are many synapses (i.e., "layers") removed from the direct visual input (Desimone & Ungerleider, 1989; Van Essen & Maunsell, 1983; Maunsell & Newsome, 1987). Furthermore, these neurons are also many synapses removed from motor output centers which can use visual object information to guide action. Thus, a good model of learning in the neocortex should be capable of solving difficult problems using multiple layers of processing units (a.k.a., *deep networks*).

While the error backpropagation algorithm does enable the use of multiple layers of hidden units, it turns out in practice that these additional hidden layers do not usually improve performance, and typically lead to longer training times. As discussed in the introductory chapter, there are reasons to believe that LEABRA might perform better than standard backpropagation in learning over multiple hidden layers. This is because the self-organizing learning in LEABRA will lead to the development of useful representations even in the absence of useful error signals (e.g., as has been seen on previous tasks where the error-driven learning component has been removed, and the network showed substantial, though not entirely successful, learning). This relative independence from error information is important because it is often the case that error signals are not very informative after they have been passed back over several hidden layers in a deep network. Thus, a LEABRA network can develop useful representations even with unreliable error signals, while a purely error-driven algorithm must develop the representations themselves on the basis of these error signals alone.

The issue of learning in deep networks is explored in this chapter in the context of the "family trees" problem of Hinton (1986). This is a multiple-relation task, as described in the taxonomy of tasks in Chapter 5, where the objective is to answer questions about the relationships between differ-

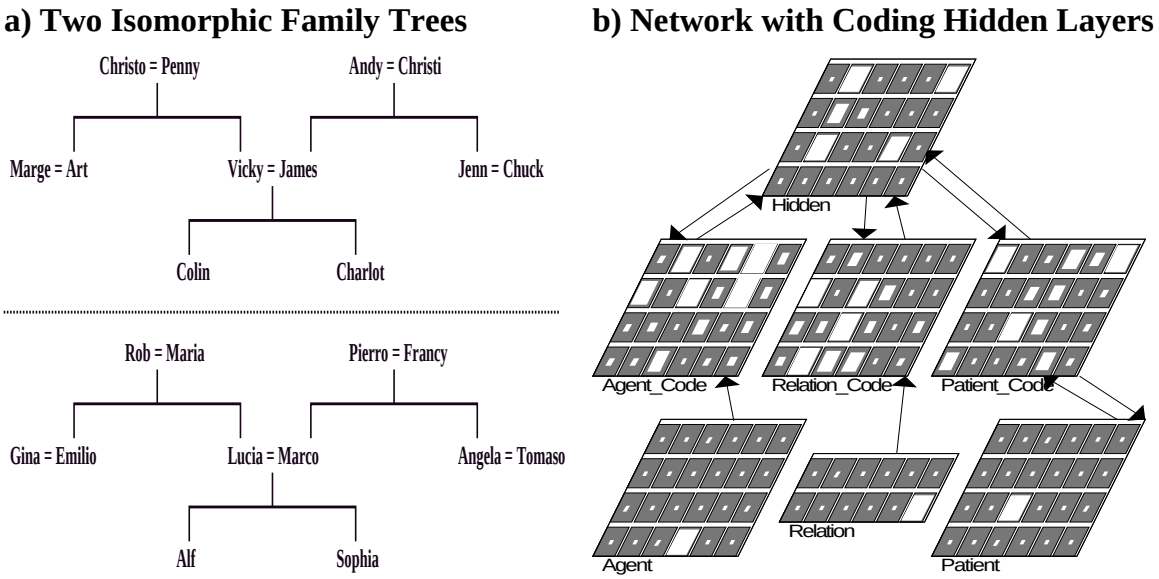

Figure 7.1: The family trees problem and network. In this network, the individual family members are represented both in the Agent and Patient layers as individual units. A version with distributed representations was also used. The code layers are supposed to develop activity patterns which capture the functional similarity of related people, so that the mapping performed by the central association hidden units is easier.

ent people in two isomorphic families. The reason it requires a deep network is that the people are represented by orthogonal input patterns, and the network has to discover an internal representation for them over a set of "encoding" hidden units, which then form the basis for solving the multiple-relation task in a central "association" hidden layer. Thus, this task displays the desired characteristic of benefitting from multiple layers of hidden unit representations.

The Family Trees Problem

Figure 7.1 shows the family tree diagram and the architecture of the network used. There are 12 people in two isomorphic families. The relationships of husband, wife, father, mother, son, daughter, brother, sister, uncle, aunt, nephew, and niece are represented. Individual training and testing patterns are produced by the triple of the agent, relationship, and patient based on the family tree (e.g., "Charles - Wife - Penny"), resulting in a total of 104 such patterns. Following Hinton (1986), 100 of these patterns were used for training, and 4 for testing. The four testing patterns were chosen to be well spaced and involve central people in the trees, since they have the highest density of information in the training set. They were: "James - Wife - Vicky", "Lucia - Father - Roberto", "Angela - Brother - Marco", and "Christi - Daughter - Jenn." For most cases, both training and testing involved the presentation of the agent and relation pattern, as a prompt for the network to produce the corresponding patient pattern. The final section explores the task where any two patterns are presented, and the network is trained to produce the correct third one.

There were three different types of input patterns used: orthogonal localist (as in Hinton, 1986),

random distributed, and feature-based. In the orthogonal localist case, there were 24 agent and patient units, one for each person, and 12 relationship units, one for each relationship. In the random distributed case, there were 25 units in the agent, patient, and relation layers, with 6 active units chosen at random to represent each person/relationship, with a maximum overlap of 2 active units with any other pattern. Note that it is not easy to represent multiple people over the same set of units using a distributed representation, so in this case, the few patterns which require multiple patient people to be activated (i.e., the multiple aunts and uncles) are encoding using only one of them. In the feature-based case, there were 5 feature groups, including the age of the person (old, medium, young), their nationality (English, Italian), their sex (M/F), and which branch of the family tree a person belonged in (left, middle, or right). These features are like those which the hidden layers are supposed to discover, and are used to illustrate how easy the mapping task is when these features are explicitly present in the training environment.

The networks had 60 hidden units in the coding and association hidden layers, whereas Hinton (1986) used only 6 units in the coding layers and 12 in the association hidden layer. While the number of hidden units is considerably larger than is necessary for BP, AP, and CHL to learn this problem, LEABRA appears to require at least 45 units to learn the problem due to its activity constraints. The larger number of units was used to ensure that LEABRA also had more units than was necessary to solve the problem. The number of hidden units in BP was varied to explore the importance of this variable on learning time and generalization performance in these other algorithms, as described below. In general, learning speed increased with the number of hidden units, and the relationship with generalization was unclear. In addition, simpler networks with only the central association hidden layer were run, in order to determine the effect of the intermediate encoding layers on performance in this task. In BP, CHL, and AP networks, a base learning rate of .1 was used (.01 resulted in very slow learning in this task). In BP, a faster learning rate of .39, which was found in Chapter 2 to be optimal for the standard version of this task, was also tested. For CHL, the optimal learning rate was the .1 value used here. However, a slower .05 learning rate was necessary to learn the random distributed inputs case for CHL. LEABRA networks always used the standard .01 learning rate. In LEABRA networks with the orthogonal localist inputs, the learning threshold for the input/output layers was set to .02 instead of the standard .1, as is discussed in the Appendix for layers with very sparse activity levels. In the random distributed inputs case, the weighting factor for the information-preservation component of the MaxIn associative learning rule was set to .1 instead of the standard .25, for reasons that are discussed below. All other parameters were standard.

Feature-Based Inputs and Task Difficulty

The feature-based input patterns provide a way of determining how difficult the input/output mapping task is even when the representations are systematic. Since this establishes an important baseline level of performance for this task, it will be described first. Because the input/output patterns in this case reflect the kinds of systematic representations over the encoding hidden layers that are

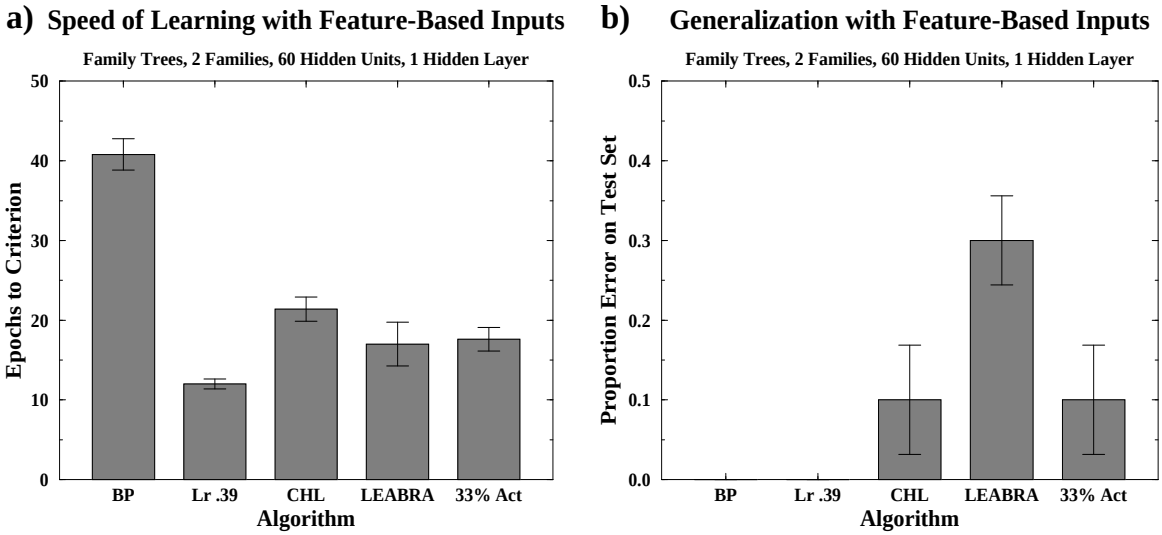

Figure 7.2: **a)** Learning speed and **b)** Generalization using feature-based inputs and a single hidden layer. **Lr .39** is BP with the "optimal" learning rate of .39. **33%** is LEABRA with 33% hidden layer activity level instead of the standard 25%, which improves generalization significantly.

supposed to develop with training in the standard model, the encoding hidden layers were not used in this case. Thus, the mapping was performed by a single hidden layer. Based on the results for training time and generalization performance in this version of the task (shown in Figure 7.2), it does not appear that learning the input/output mapping component of this task is very difficult, and once learned, generalization can be perfect (at least in the BP networks). This makes sense, since there are not that many people or relationships to encode, and it is a very systematic mapping using these features. Further, note that CHL learns faster than BP for the same learning rate in this version of the task, but this will not remain true for the deep networks.

Perhaps the most interesting aspect of the results for this simplified version of the task is the relatively poor generalization performance of LEABRA. This will be discussed further in the context of the generalization results on the other versions of the task below. However, it should be noted that there is a possible explanation for this result in terms of the specific properties of the feature-based inputs. This explanation has to do with the generally poor performance of LEABRA when the input/output patterns have a higher activity level (i.e., above 30%). In this task, 40% of the agent and patient units are active on each pattern. A similar problem was apparent in a version of the combinatorial environment task studied in the previous chapter which had 40% activity levels, and has also been observed on other tasks not reported in this thesis. For more details see the Appendix. One hypothesis about how to solve this problem is to try to more closely match the activity levels of the input/output and hidden layers. Thus, the *33%* hidden layer activity condition shown in the figure was run, which did improve generalization, but only to the level of CHL. In previous tasks, LEABRA has performed as well if not better than BP. Thus, it is possible that there is a more general problem with LEABRA in generalization on this type of multiple-relation task, as will be discussed

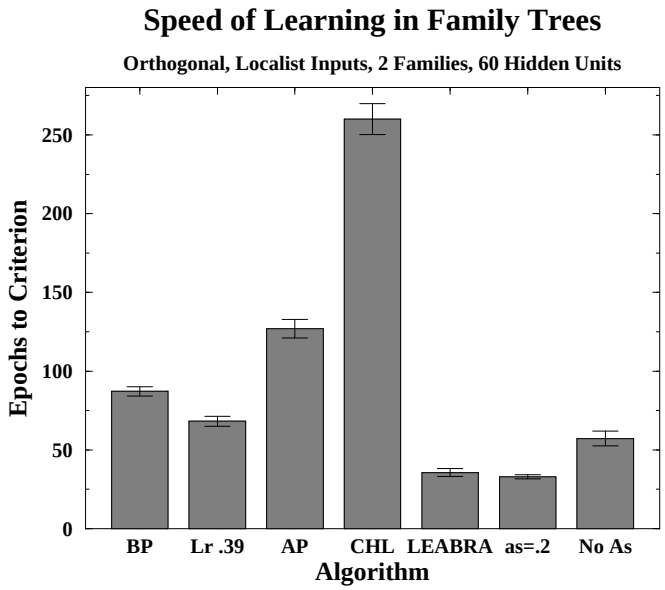

Figure 7.3: Learning speed for family trees task with orthogonal localist inputs. For BP, AP, and CHL, the fast learning rate of .1 was used, while LEABRA used the standard .01 learning rate. The **BP Lr .39** case was BP with 60 hidden units at the optimal learning rate (.39) for this task as found in Chapter 2. Note that .1 was the optimal learning rate for CHL. **as=.2** is LEABRA with associative learning strength of .2, and **No As** is LEABRA without associative learning (ReBel + CHL).

below.

Orthogonal, Localist Inputs and Symmetry

The important challenge in the deep network version of this task is the development of encoding representations of orthogonal, localist inputs that capture the same information used in the simplified feature-based version of the task. The particular choice of the orthogonal localist input representations transforms this otherwise simple task into a very difficult one. The original results reported by Hinton (1986) required around 1,500 epochs of training in a feedforward backpropagation network with optimized learning rate and momentum parameters. I was able to replicate the difficulty of training these networks using the same numbers of hidden units used by Hinton (1986), as is shown in the results presented later. However, when using a larger number of units with the on-line (per pattern) form of weight update, which has been used in all of the previous simulations reported in this thesis, learning time decreased by more than an order of magnitude. The reason for this improvement with more units and on-line learning is probably due to the breaking of the symmetry of the error signals in this task as a result of the random variation in the order of pattern presentation, and the larger random sample of unit weights. Phenomenologically, in batch mode learning, there is a long, flat plateau in the error surface, the traversal of which consumes most of the training time. In on-line learning, this plateau is diminished.

However, even in the on-line mode, the symmetry of the error signals contributes to the difficulty of learning this problem. This symmetry is due to the completely orthogonal, localist output

units, which are sparsely and relatively uniformly activated over the training set. Because the predictability of the output units depends on an interaction between the relationship and agent inputs, the input-output mapping is not predicted by either of them individually, resulting in an initially very ambiguous and symmetric pattern of up/down error signals for each output over the training set. Until the interaction between the agent and relationship inputs begins to be encoded, the error signals lead to this up/down "thrashing" behavior. This would predict that the use of random distributed patterns would help to break the symmetry through the random overlaps between different patterns, and should lead to even faster training times, as is explored below.

Thus, rather than the mapping task itself, it is the ambiguity and symmetry of the error signals in the orthogonal localist version that gives this task its difficulty. This source of difficulty interacts with the depth of the network, which compounds the problems with the error signals. Thus, both network depth and the ambiguity and symmetry of the error signals serve as a proxies for the problems with purely error driven learning in more difficult tasks, in even deeper networks. LEABRA should avoid these problems to some extent by developing representations independent of the error signals by virtue of its self-organizing learning component. Figure 7.3 shows that, as hypothesized, LEABRA learns this task significantly faster than any of the purely error-driven algorithms. Note that this is the case despite the fact that LEABRA is using a learning rate an order of magnitude smaller than the other algorithms. If BP is run with a .01 learning rate, it takes 548, SEM 20.6 epochs to learn. Thus, whereas the purely error-driven algorithms have to be pushed to somewhat extreme learning rates, which will have negative consequences (e.g., for interference with existing knowledge in the network, McClelland et al., 1995), LEABRA naturally learns this task rapidly. Several other important results are evident in this figure:

MaxIn associative learning improves learning speed: As was the case with the generalization results from the other tasks, the relatively fast learning of LEABRA is at least partially dependent on the MaxIn associative learning, since LEABRA without this (*No As* in the figure) learns more slowly, and increasing the strength of the associative component to .2 (over the default of .1) results in slightly faster learning. This is consistent with the idea that the self-organizing learning of useful representations contributes to LEABRA's better performance in deep networks. Note that the increase in learning speed associated with MaxIn is not simply due to a larger effective learning rate, since increasing the learning rate (over the standard .01 for the above results) actually led to *slower* learning (lrate of .015: 55.8, SEM 4.99 epochs, lrate of .02: 76.2, SEM 9.07 epochs, compared to 35.6, SEM 2.51 for lrate .01).

ReBel activation constraints improve learning speed: If the *No As* case (ReBel + CHL) is compared to the CHL condition, it is clear that the ReBel activation constraints are making a large contribution to the learning speed in LEABRA. This was also the case for generalization performance in the other tasks studied, and is probably due to the specialization of representations (entropy reduction) as a result of the competition. The resulting differentiation of hidden unit representations breaks the symmetry of the error signals. Further, it is possible that the damping effect of ReBel

makes a deep network somehow more "stable" than an unconstrained network would be, and that this stability contributes to better learning (e.g., by analogy to the balancing of poles presented in the introduction).

Interactivity impairs learning speed: The relationship between BP, AP, and CHL in this task, which is similar to that found in the other tasks in generalization performance, indicates that the gradient of increasing levels of interactivity over these algorithms has a concomitant effect on learning speed. Thus, BP is the fastest, with AP performing intermediate between it and CHL, which is the slowest. This pattern is the opposite of what was observed in terms of learning speed in the previously studied tasks (though learning speed was not the focus in these cases, and was not thus reported), and importantly on the feature-based version of the task as reported above. See Chapter 2 for a discussion of this issue and some detailed learning rate results for shallow networks. The detrimental effect of interactivity can potentially be explained in terms of the relative instability of these networks — an interactive network with multiple hidden layers is even more sensitive (i.e., susceptible to butterfly-effect kinds of nonlinearities) than a shallow interactive network, since the opportunity for non-linear interactions is greater as the number of layers increase. Thus, extra training time is necessary to form the stable attractors necessary for learning the training patterns. This explanation makes an interesting prediction — if the CHL network is spending extra training time stamping out spurious attractors that interfere with the reliable learning of the training items, this might actually result in relatively *better* generalization performance. This prediction is tested below.

Self-organizing learning only (No Err) cannot solve this task: As was the case on previous tasks, LEABRA without the error-driven CHL learning component cannot solve the family trees task. The initial sum-squared-error (SSE) was 181, and the network got as low as 63, but, as was the case previously, the number of patterns learned to criterion revealed less complete learning, with 70 patterns incorrect out of 100 being the best performance. As before, this confirms the importance of error-driven learning for being able to actually learn tasks.

GausSig improves learning speed: (not shown in figure) The use of the GausSig likelihood function (see Chapter 4) was important for the fast learning observed in LEABRA. The above results are all with GausSig and the standard k_{gauss} parameter of 2. When the sigmoidal likelihood function was used instead, learning speed decreased to 354, SEM 248 epochs (compared to 35.6, SEM 2.51 reported above). On the other hand, when k_{gauss} was increased to its maximal value of 4, learning speed increased to only 26.4, SEM 2.22 epochs. Thus, while other tasks showed only moderate improvements associated with using the GausSig function, this task showed a significant one. This may indicate that GausSig is more important for deep networks, where there are multiple hidden layers with more graded activity values than the simple binary input states. The effects of GausSig compared to the sigmoid are most apparent with intermediate sending activation values.

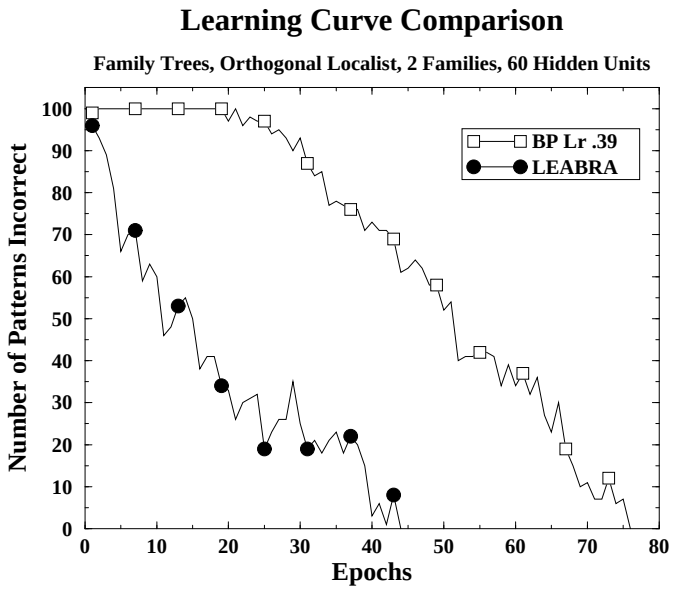

Figure 7.4: Learning curve from a single training run of BP with learning rate of .39, and standard LEABRA, using orthogonal localist inputs and 60 hidden units. BP has an initial plateau due to symmetry of error signals, while LEABRA learns rapidly from the start.

Factors Contributing to Fast Learning in LEABRA

The learning speed advantage of LEABRA is hypothesized to derive from its ability to develop useful representations early on in learning as a result of its self-organizing learning component. Aside from the fast overall learning speed and other results presented above, two additional forms of evidence are consistent with this hypothesis. One is the shape of the learning curves over time, and the other is a cluster analysis of the hidden unit representations over time.

Figure 7.4 shows learning curves for BP and LEABRA, indicating that, as described above, BP has an initial plateau where it is getting essentially nothing correct, and the error signals are not resulting in the development of useful representations. In contrast, LEABRA shows rapid learning from the very start. This can be attributed to the self-organizing development of useful representations independent of the otherwise not very effective error signals. In addition, it should be noted that there is a bootstrapping effect here, since once somewhat useful representations have been developed (i.e., representations that enable partially correct performance), this causes the error signals to be much more informative, since they lose their symmetry and begin to provide a discriminative signal for learning.

A more detailed view of the differences between learning in BP and LEABRA can be obtained by performing a cluster analysis of the hidden unit similarity structure (over the central hidden layer) as the network learns. This analysis is based on the OR (max) of the hidden unit activity values over all training patterns in which the given agent (as indicated in the plot) appeared. Note that one of the testing patterns has Christi as an agent, and is thus missing from the training set over which the cluster plots were generated, resulting in the odd results for this case. Figure 7.5 shows cluster plots

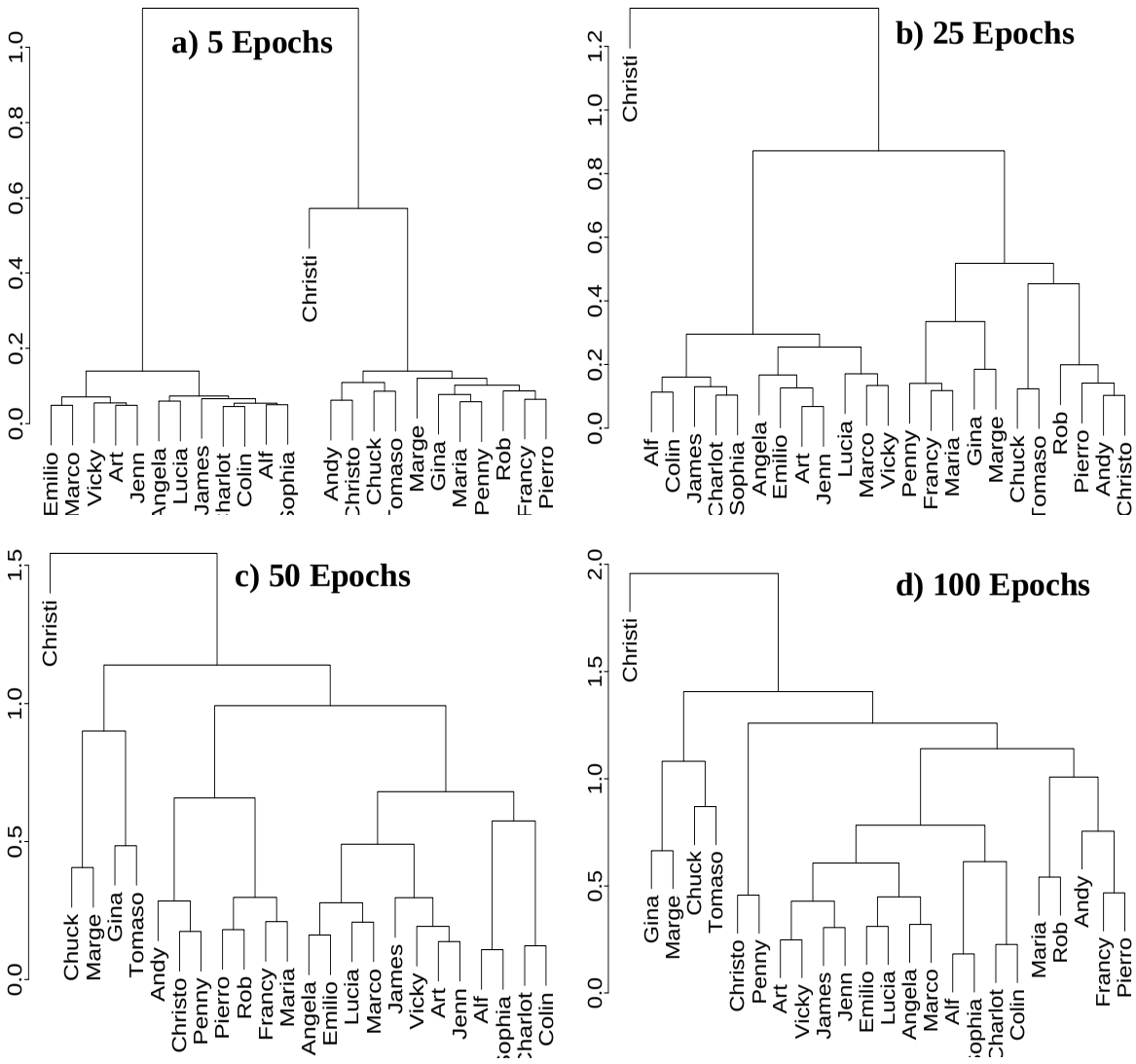

Figure 7.5: Development of representations in BP over time in terms of a cluster plot of the similarity structure of hidden unit representations (central hidden layer) for the Agents shown. Clustering was computed on the OR (max) of the activity values for training patterns in which the given agent appeared. Early in training (5 and 25 epochs), there is relatively little differentiation.

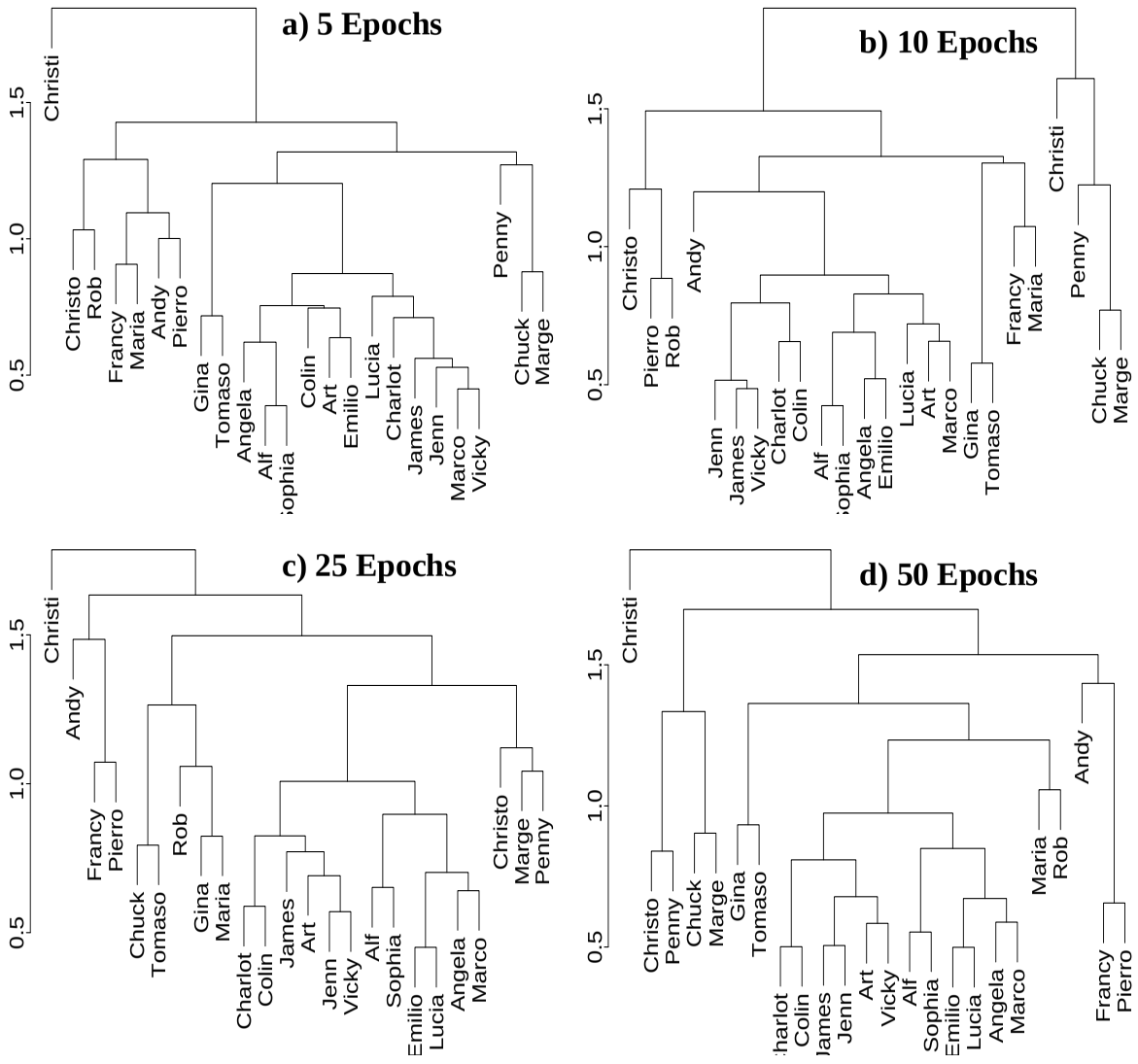

Figure 7.6: Development of representations in LEABRA over time as in the previous figure. In contrast to BP, considerable structure is evident after just 5 epochs of training.

for 4 different points in training for the BP network whose learning curve was plotted in Figure 7.4, and Figure 7.6 shows cluster plots for the LEABRA network (note the epoch when each plot was taken differs between the two figures due to differences in rate of representation development).

Perhaps the most obvious difference between the two algorithms is that the BP network shows a gradual development of differentiation, starting from almost no structure at 5 epochs, while the LEABRA network has developed clearly differentiated hidden unit representations as early as 5 epochs. This early differentiation is exactly what would be expected from the self-organizing learning (activity competition and Hebbian associative learning) in LEABRA, and is consistent with the idea that this plays an important role in rapid learning in deep networks.

A more detailed analysis of the development of structure in these cluster plots is possible. One interesting result is that the final clusters for both BP and LEABRA are remarkably similar — both contain a cluster for 2nd and 3rd generation people (with the exception of the "outsiders" in the 2nd generation), within which the English and Italians are divided, and the terminal clusters represent sibling pairs. The 1st generation are in marriage pairs (except for Christi and Andy), and the outsiders are either grouped by nationality or homologous position. Elements of this final organization are evident early on in both BP and LEABRA clusters — but LEABRA exhibits this early structure with much greater differentiation between the different clusters than the BP network. The final level of differentiation is similar for both BP and LEABRA.

Generalization and Parameters Affecting It

One of the manipulations that was necessary to make any of the learning algorithms learn this task in a relatively small number of epochs (order 100's instead of order 1000's) was to increase the number of hidden units in the coding and association hidden layers beyond the 6 (coding) and 12 (association) units used by Hinton (1986). However, this increased number of units might impair the generalization performance in this task, since the original motivation for using so few units was to introduce a *bottleneck* in the coding layers that forced the network to encode the items efficiently and systematically, leading (in theory) to better generalization. The reliance on such a bottleneck for good generalization performance is not particularly biologically plausible, given the vast numbers of neurons in the neocortex. Thus, it is of interest to determine the effects of hidden layer size on generalization (and learning speed) performance.

Figure 7.7 shows that, contrary to the expectation, there were no apparent effects of creating a bottleneck in the coding layers on generalization performance. However, there were dramatic effects of the bottleneck on learning speed, with the 6 coding, 12 association case used by Hinton (1986) (6/12 in the figure) learning an order of magnitude slower than the case with 60 hidden units. Thus, there is no apparent tradeoff between learning speed and generalization. Nevertheless, the generalization performance is not very impressive. While Hinton (1986) reported successful generalization on 3 out of the 4 testing items (generalization error of .25) in the one network he tested, I know of no existing systematic study of generalization rates in this task. Also, I have found that by including in the test-

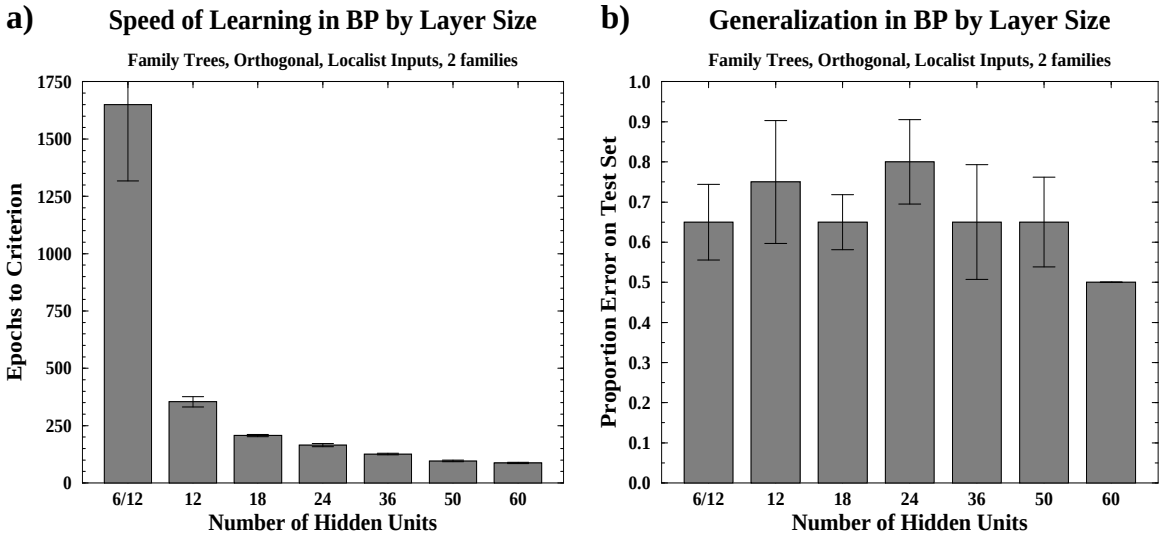

Figure 7.7: Effect of number of hidden units on **a)** Learning speed and **b)** Generalization performance in BP. The **6/12** case is 6 coding and 12 association hidden units, as used in the original Hinton, 1986 model. The remainder are the numbers of both coding and association hidden units. Learning rate was .1 in all cases, as 6/12 case was not able to learn to criterion at the .39 "optimal" learning rate.

ing set different numbers of triples involving the relations aunt, uncle, niece and nephew, which have only one corresponding patient per family, the generalization score can be improved considerably. However, this is not a very good test of abstract generalization, since it can be solved by the simple memorization of the relation with its corresponding patient, and requires only the disambiguating bit of which family it is. Thus, the testing set used for these results has only central people which participate in many different relationships with different people, for which a truly systematic and abstract encoding must be developed in order to generalize properly. The results from the feature-based inputs show that if the network was actually producing systematic representations over the encoding units, generalization should be perfect with the same training and testing items used here.

One other potentially important difference between the networks used by Hinton (1986) and those used here is form of weight updating — Hinton (1986) used batch mode, while all the networks reported here use on-line learning. It is possible that networks trained with batch mode will exhibit better generalization by virtue of each weight update reflecting the entire error gradient over all patterns, instead of just the local gradient for one pattern. As discussed above, the use of on-line learning results in faster learning than batch mode, as a result of the symmetry-breaking influence of the random order of pattern-wise weight updates. Thus, there may be a cost in generalization associated with this faster learning. To test this possiblity, and to replicate most closely the conditions used by Hinton (1986), networks with 6 encoding and 12 central hidden units were trained with batch mode learning. A schedule for learning rate and momentum parameters (suggested by G. E. Hinton, personal communication) had to be used in order to obtain reliable convergence in batch mode. This was .0025 learning rate, .5 momentum for the first 20 epochs, followed by .01 learning rate, .95 mo-

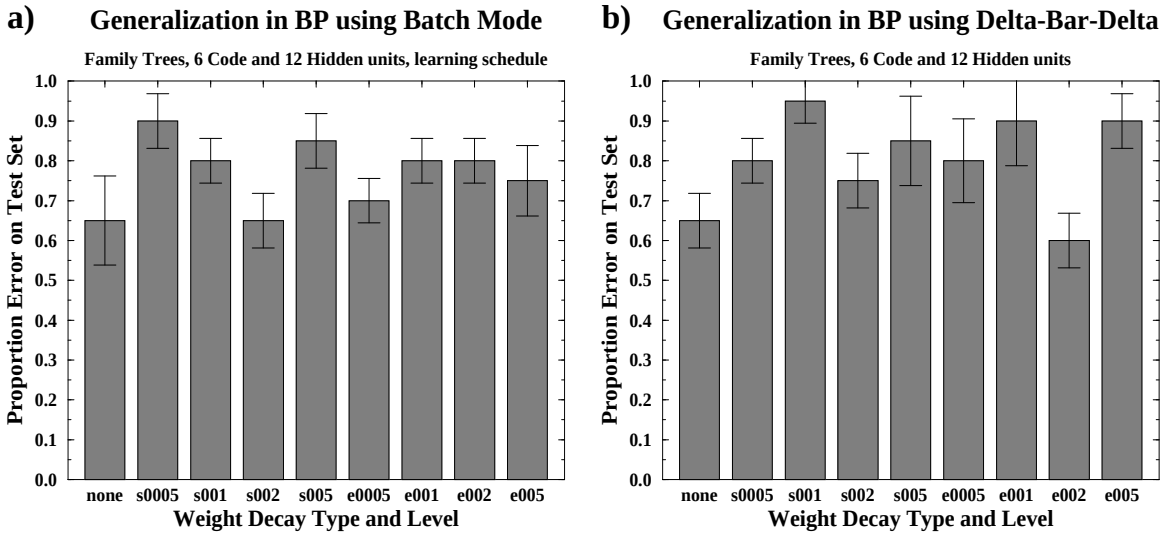

Figure 7.8: Generalization performance for batch mode learning, using **a)** a learning rate schedule and **b)** delta-bar-delta learning rate adaptation. **s** weight decay is simple weight decay (SWD), and **e** weight decay is weight elimination (WED). Results do not represent an improvement over the on-line case.

mentum for the remainder of learning. As an additional test, delta-bar-delta learning rate adaptation, which involves the use of separate learning rate parameters for each connection (Jacobs, 1987), was also used to train batch-mode networks, with a learning rate of .01 and no momentum. For both types of batch-mode learning, a range of weight decay strenghts were used for both simple weight decay (SWD) and weight-elimination weight decay (WED). The results are shown in Figure 7.8, and indicate that the relatively poor generalization observed previously in the on-line case is also characteristic of batch mode learning on this task. Further, no amount or type of weight decay appeared to result in a generalization improvement.

The generalization performance of the various other algorithms, all with 60 hidden units, is shown in Figure 7.9. As was the case with the BP networks, there is no clearly interpretable pattern of results that emerges from this figure. Note that the .5 generalization of the BP network is probably a fluke, since the version with the faster learning rate, which did not have a systematic effect on other networks with different numbers of hidden units, generalized worse. Perhaps the most notable result is that, in contrast to all the other tasks studied, the CHL networks performed at roughly the same level as the other algorithms. This relative improvement may be due to the CHL network having eliminated many of the potential spurious attractors over learning, due to the increased sensitivity of a deep interactive network, as suggested above. Also, as was the case in the feature-based version, LEABRA appeared to generalize worse than the other algorithms, as will be discussed further below. Finally, there is a weak indication that associative learning improves generalization performance in LEABRA, since the case with more associative learning (.2) generalized slightly better, and the case without any associative learning at all (No As) generalized slightly worse. However, this difference is clearly not very substantial.

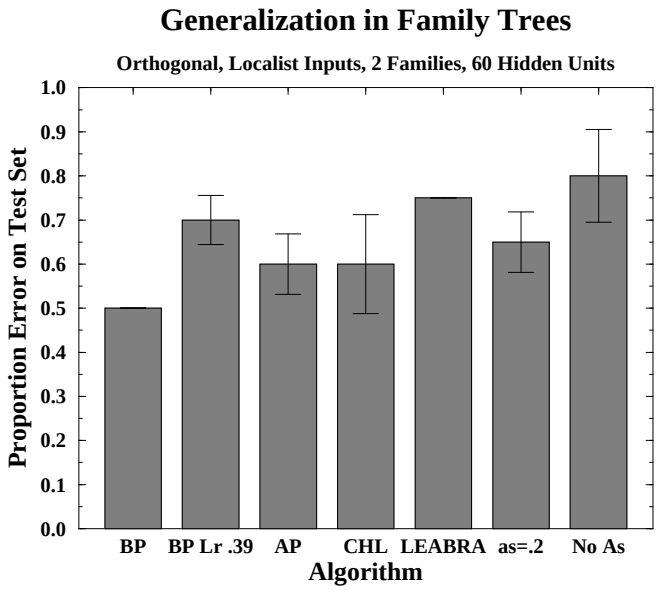

Figure 7.9: Generalization performance for various algorithms tested. The algorithms are as described in Figure 7.3.

One possible explanation for the relatively poor generalization performance in the family trees task is that there are not enough training items to establish a basis for generalization. This idea can be tested by increasing the number of families in the training set. Thus, up to two additional isomorphic family trees (a German and a Japanese family) were created, and represented by simply introducing more localist units in the input/output layers, resulting in patterns for 3 and 4 total families. For both the 3 and 4 family sets, eight (instead of the 4 used in the 2 family version) patterns were drawn at random for the testing set, until the entire group of 8 patterns did not contain a single niece or nephew pattern, for the reasons described above. BP networks were then trained on the 3 and 4 family versions, and generalization measured on the corresponding testing set. The generalization results, shown in Figure 7.10, indicate that while there is some improvement with increasing numbers of families, this improvement is not dramatic — the absolute level of generalization performance even with 4 families is still considerably worse than the perfect performance obtained with the feature-based version of this task. Thus, the considerable additional computational expense of running the interactive algorithms on these larger family sizes did not seem justified, and was not performed.

Finally, the effect of weight decay on generalization in BP with different numbers of hidden units was assessed. Figure 7.11 shows that, while it did have a positive effect in the large (60 hidden unit) network, it did not improve performance in the small 6/12 hidden unit network used by Hinton (1986) (as was observed in the batch mode results presented earlier). Also, using a larger decay parameter (.005 instead of .002) prevented the networks from learning at all. The use of weight decay, which is an adapting constraint, reliably slowed the learning speed of the networks (68.2, SEM 3.23 without weight decay, 86.4, SEM 9.91 with .002 weight decay).

It seems safe to conclude, based on these results, that none of the networks does a very good job

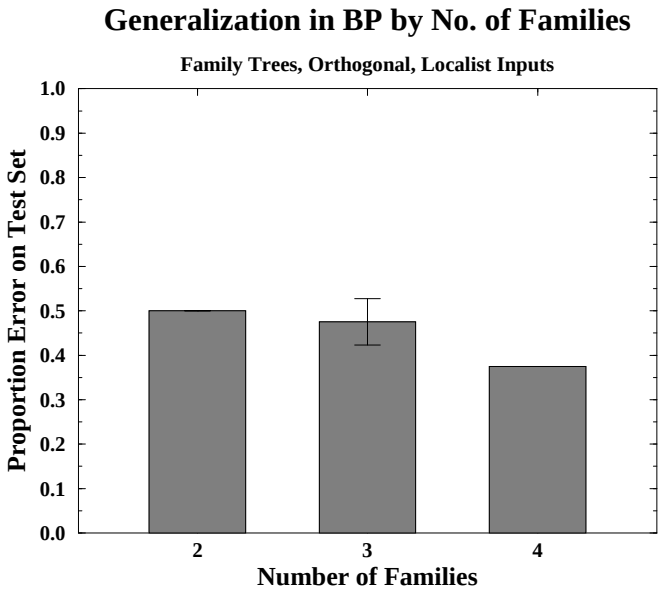

Figure 7.10: Effects of the number of families (training set size) on generalization performance in BP. While there appears to be some improvement in generalization for the 3 and 4 family sets, it is not clearly monotonically increasing with the number of families, and does not represent a substantial absolute improvement.

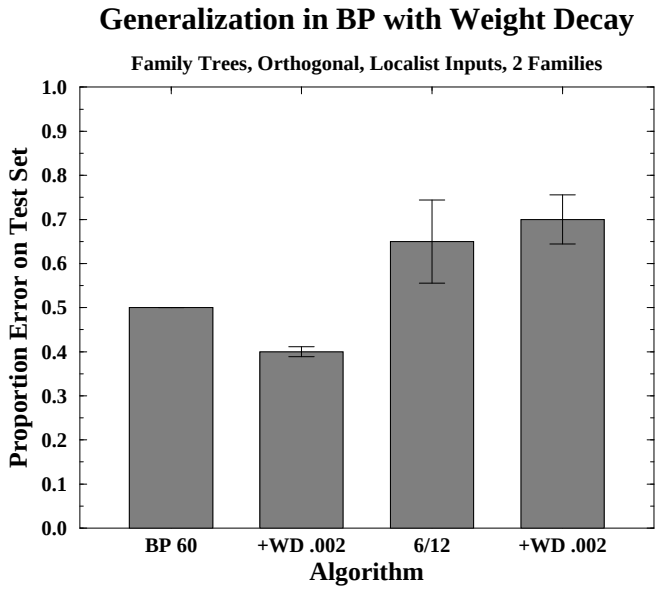

Figure 7.11: The effects of weight decay on generalization performance in BP. **+WD .002** is the previous condition plus .002 weight elimination. Note that .005 weight elimination prevented the networks from reaching criterion on training in all cases. **BP 60** is the standard network with 60 hidden units. **6/12** is the original small 6 coding, 12 association hidden units.

at a more thorough test of systematic generalization than the one performed by Hinton (1986). In the case of LEABRA, it is possible that the multiple-relation aspect of this task contributed to impaired generalization, considering its relatively poor performance even in the simplified feature-based input task as reported above (compared to its relative generalization advantage on other tasks studied in previous chapters). It is possible that the modulation of the agent/patient mapping that is supposed to take place via the relationship inputs requires that they have a more multiplicative, rather than additive, effect. Since the same input pattern must enter into several different relationships depending on the state of the relationship input, the mere addition of this input into the overall net input seems ill suited for its modulatory role. Instead, a more multiplicative effect, like that performed by the gating unit in the mixtures of experts framework (e.g. Jacobs et al., 1991) might work better.

Rather than simply viewing the relatively poor generalization of LEABRA and the other algorithms on this task as a failure, it can instead provide computational insights which might be useful in understanding why the brain is structured in the way it is (c.f. McClelland et al., 1995 for how the "catastrophic interference" failure of neural networks led to an understanding about the possible division of labor between the hippocampus and neocortex in learning and memory). Thus, it is interesting to speculate that the neocortex might have developed a specialization for contextualizing information in a manner similar to that required by multiple-relation tasks like family trees. There is evidence that the prefrontal cortex (PFC) is responsible for representing the context necessary to disambiguate the meanings of ambiguous words, for example, and its hypothesized role in controlling the activities of neurons in posterior cortical areas is similar to that of the relationship inputs in this task (Cohen & O'Reilly, 1996; Cohen & Servan-Schreiber, 1992; Cohen, Dunbar, & McClelland, 1990). It is possible that this specialization exists in part to provide a more suitable mechanism for selecting among different possible input/output mappings in a way that generalizes better than simple homogeneous networks like those tested here. At this point, this remains a topic for future inquiry.

Random Distributed Input Representations

As mentioned above, one important consequence of using random distributed input patterns instead of the orthogonal, localist ones is that their random overlap breaks the symmetry of error signals during training, and thus should accelerate the rate of learning for purely error-driven learning algorithms. However, this random overlap could conceivably make the task more difficult, as the network also has to learn to ignore this random overlap and encode inputs according to their systematic relationships. The results for this case are shown in Figure 7.12, which confirms that the net effect is an increase in learning speed for the purely error-driven algorithms. However, these algorithms consistently generalized worse in this case than in the orthogonal localist one. In contrast to this pattern of results, LEABRA learned more slowly, but retained roughly the same level of generalization performance compared to the orthogonal localist case.

One interpretation of this pattern of results is that the purely error-driven algorithms (BP, AP,

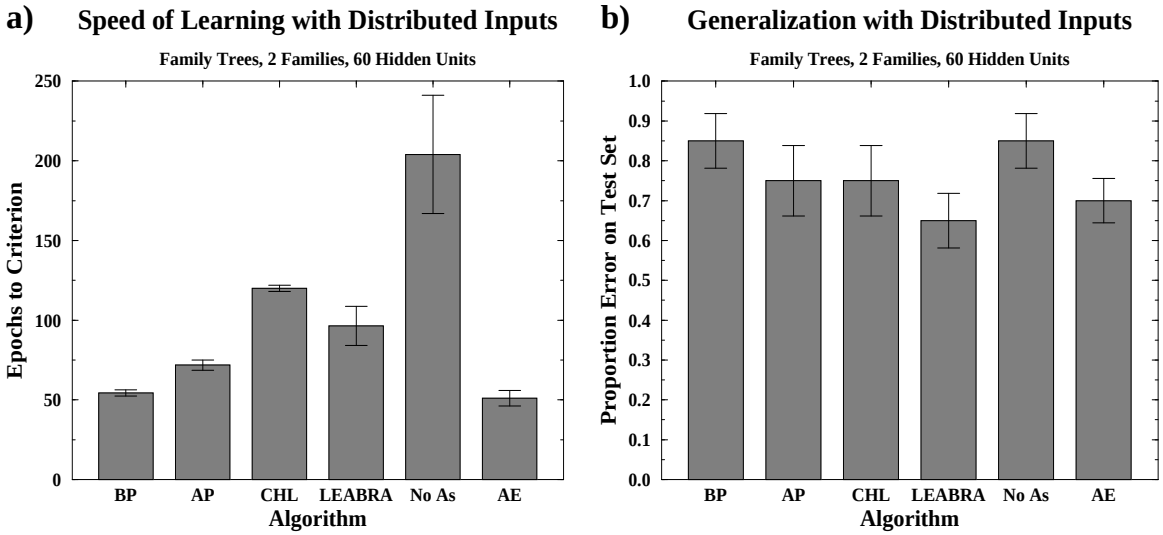

Figure 7.12: **a)** Learning speed and **b)** Generalization performance using random, distributed input patterns instead of the local, orthogonal ones. Algorithms are as in previous figures, with the addition of **AE** which is the LEABRA auto-associator model.

CHL) are representing items at least somewhat according to their random similarity, but that this provides a sufficient basis to learn the mapping task. However, since these representations are not terribly systematic, generalization suffers. In contrast, it appears that LEABRA did not represent the patterns as much according to their random similarity (given its generalization performance), but the process of developing more systematic representations in the face of the random overlap required more training time. The role of associative learning appears to be quite important in this process, since without it (*No As* in the figure), the network took significantly longer to learn, and generalized worse. Associative learning is likely forming associations based on the similarity structure over the entire input/output pattern ensemble, as described previously. By comparing the No As (ReBel + CHL) case with the CHL network in this random distributed version *vs* the orthogonal localist one, it appears that the ReBel activation constraints are less advantageous in this case. This could be due to the reduced representational capacity of a kWTA system compared to one without activity constraints.

This task provides an interesting insight into the tradeoff between the information preserving soft-competitive learning (SCL) and the entropy reducing zero-sum Hebbian (ZSH) components of the MaxIn learning rule. Because the input and output patterns are random, it seems reasonable that it might be better to reduce the drive to preserve information about them, and emphasize entropy reduction more. Indeed, this appears to be the case, since LEABRA learned this task better with a weighting term of .1 on the SCL MaxIn component than with the standard .25 value (which had an average learning speed of 146 epochs compared to 96 for the .1 value shown in the figure).

Finally, the figure shows results from the auto-encoder version of LEABRA, which was not run in the orthogonal localist case because there is no real structure to be represented in individual input

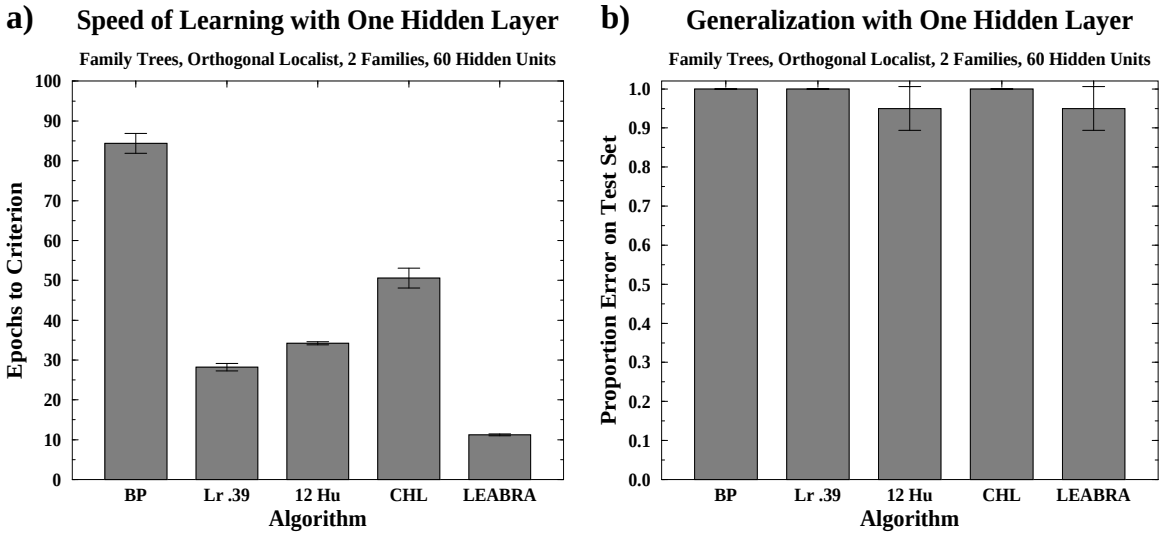

Figure 7.13: **a)** Learning speed and **b)** Generalization using orthogonal, localist inputs and a single hidden layer with 60 units (unless otherwise noted). **Lr .39** is BP with the optimal .39 learning rate. **12 Hu** is BP with 12 hidden units and the .39 learning rate.

patterns. However, in this case, there is no systematic relationship between the input pattern structure and the mapping task, which is probably why AE LEABRA did not generalize better than it did. Nevertheless, it is interesting that it did have the fastest learning time of any of the algorithms tested. Thus, the additional error-driven pressure to represent the input/output patterns probably caused the representations to develop more rapidly, but at the cost of their systematicity with respect to the input/output mapping task.

Family Trees in Shallow Networks

The results from the feature-based inputs version of this task showed that all algorithms could learn the basic mapping task in around 20 epochs in a network with one hidden layer (no encoding layers). In this section, the extent to which a single hidden layer is sufficient to learn the task even with the orthogonal input representation is explored. If it is the case that the extra encoding layers are truly facilitating the learning of this task, one might expect that a network with a single hidden layer would learn more slowly than one with the encoding layers. On the other hand, since purely error-driven algorithms typically learn more slowly in deep networks, one might expect that these algorithms will learn faster with a single hidden layer. In any case, it seems clear that generalization should be better with the use of the encoding hidden layers.

The results, shown in Figure 7.13, indicate that the additional encoding layers are largely responsible for the slow learning of this task, even with the orthogonal input representation. Every algorithm, with the exception of BP with the slower learning rate, learned at least twice as fast as the deep network version. This can be attributed to the problems associated with passing error signals back through multiple hidden layers, as described earlier. This figure also shows that, as expected, gener-

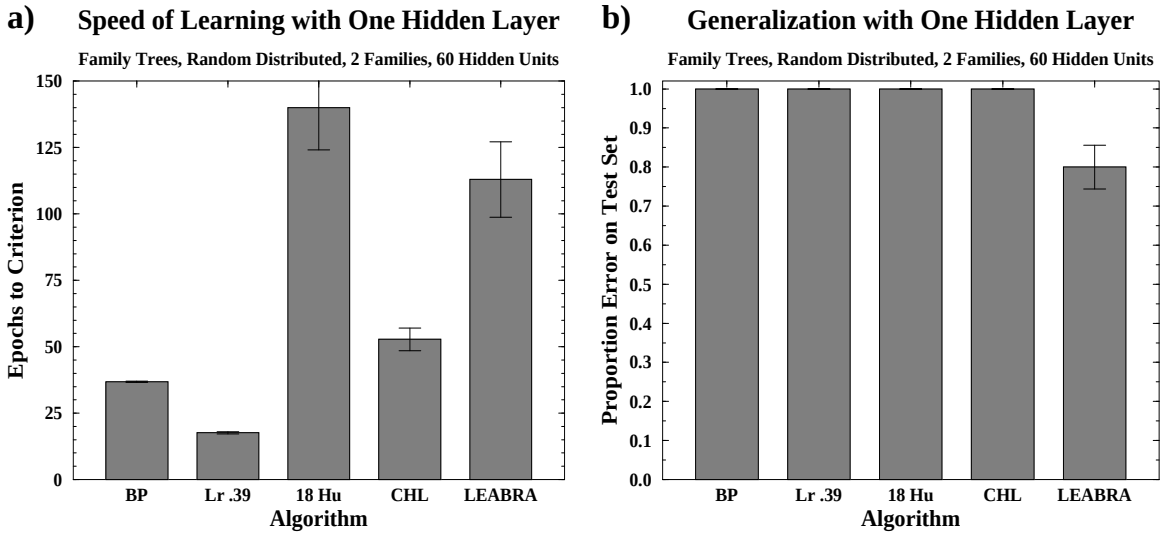

Figure 7.14: **a)** Learning speed and **b)** Generalization using random, distributed inputs and a single hidden layer. **Lr .39** is BP with the optimal .39 learning rate. **18 Hu** is BP with 18 hidden units and the .1 learning rate (could not learn with .39 learning rate).

alization suffers dramatically with only one hidden layer. However, it is interesting to note that the 12 hidden unit BP network and the LEABRA network were both capable of getting one of the four testing questions right in one out of the five networks (note that this testing pattern was produced correctly over multiple testing trials, so it was not just a random occurrence). Thus, at least some rudimentary level of generalization is possible even with only one hidden layer. It is likely that the bottleneck of 12 units was important for the BP network to develop a somewhat systematic encoding of the patterns over the one hidden layer, since this level of generalization was also observed for the 18 hidden unit case, but not for 24, 36, 50 or 60 hidden units.

Given the rapid learning speeds for the orthogonal localist inputs with one hidden layer, it is of interest to see if the random distributed inputs present a greater or lesser challenge in this context. Certainly, one would expect that generalization would be more difficult given that the random overlap will be hard to overcome with only one hidden layer. The results, shown in Figure 7.14, indicate that learning speeds were even faster in this case for the purely error-driven algorithms, except when there was a bottleneck (the 18 hidden unit case). This parallels the increased learning speed observed with random distributed inputs in the deep network, and is likely occurring for the same reason — the breaking of error symmetry due to the random pattern overlaps. In contrast with the purely error-driven algorithms, LEABRA learned almost an order of magnitude slower than with the orthogonal localist inputs. Indeed, in this case, LEABRA actually learned more slowly in the shallow network than in the deep one.

The generalization picture, also shown in Figure 7.14, is pretty much as expected, except for the remarkable results for LEABRA. Thus, both BP and CHL do not generalize at all, even with the bottleneck in BP. LEABRA, however, reliably generalized to one out of the 4 test cases in 4 out of

the 5 networks tested. Given that it was the same test case that was correct in all four networks, it appears that there is some aspect of the random correlations among the distributed patterns that led to correct performance on this test case. Nevertheless, LEABRA did generalize in the orthogonal, localist case, where there were no such random correlations, so it is possible that the contribution of this fortuitous correlation is fairly minor, but sufficient.

Flexible Access to Knowledge in Interactive Networks

If the family trees task is to be taken literally as an example of how humans can encode knowledge about the semantic properties of other people (in this case, their relationship semantics), it should be the case that this knowledge can be learned and accessed in a more flexible manner than that used in the original version of this task. In general, it should be possible to produce the third component of the agent-relationship-patient triple given any two. Thus, in addition to the standard form of this task, it should be possible to present two people to the network and have it produce the relationship that exists between these people, and similarly for it to fill in the agent slot in response to a patient and a relationship input. Of course, in order to do this, an interactive network with bidirectional connectivity is required, so that information presented on any of the input/output layers can flow in the appropriate direction to inform the answer produced at the other layer.

To test how well the networks perform on a task of this nature, the interactive algorithms (CHL, AP, and LEABRA) were trained on a version of the family trees task with orthogonal, localist inputs that were presented to any two out of the three input layers, and the target was the appropriate response on the third layer. While the mapping task is unambiguous for both the agent+relationship = patient and agent+patient = relationship cases, it is not for the patient+relationship = agent case, since, for example, the answer to the question "Jenn is the daughter of whom?" has two answers (her father and her mother). Thus, in order to maintain the one-to-one nature of the task (i.e., so that distributed representations could be used), and still disambiguate which answer was the correct one, a "sex" input/output unit was introduced for both the agent and patient layers. Thus, when probing for an agent, the sex of the desired agent was activated (e.g., female for mother, male for father in the above example). To simplify the training, the agent-sex and patient-sex units were treated as additional layers connected in the same manner as the agent and patient layers, and the network was simply asked to produce the correct answer on one out of the 5 possible input/output layers (chosen at random on each training trial) given inputs on the remaining four. Thus, this is a "rotating question" version of the task. An epoch was counted as one pass through the 100 training items, with the question asked of each item selected at random for that item. This means that only one out of the 5 possible permutations of questions for each item were trained in each epoch. The training criterion was still the standard 95% correct for all items in a given epoch. Testing was with the same 4 test cases used previously, but in all three major directions of interest (agent+relationship = patient, agent+patient = relationship, and patient+relationship = agent), for a total of 12 test cases.

The results for learning speed and generalization in this interactive, rotating question version of

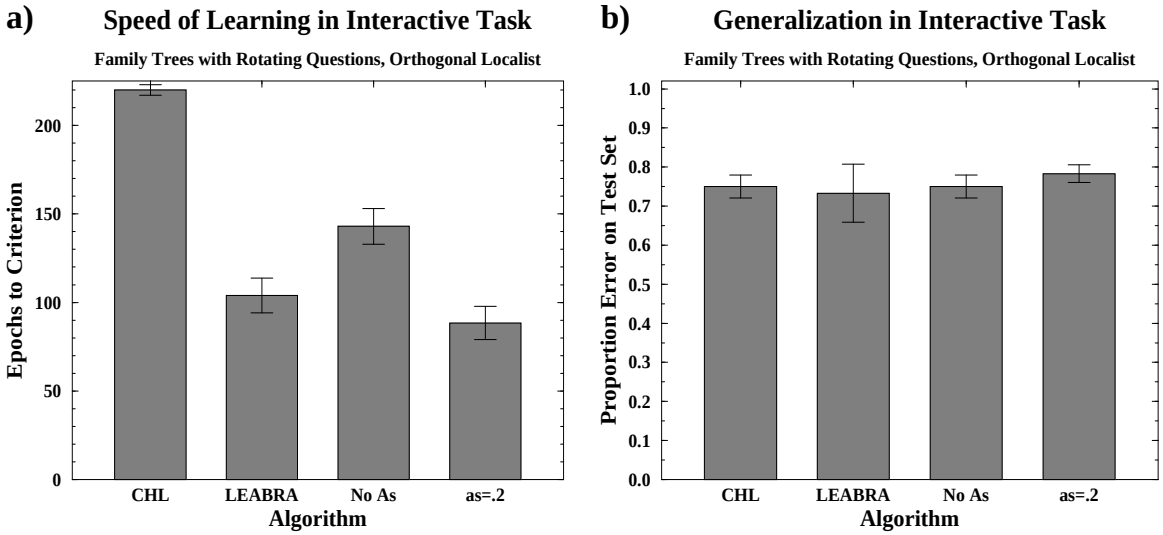

Figure 7.15: **a)** Learning speed and **b)** Generalization in an interactive version of the family trees task where the inputs and questions are rotated on each trial. **No As** is LEABRA without associative learning, and **as=.2** is with .2 associative learning.

the task are shown in Figure 7.15. Notably absent from this figure are results from the AP algorithm, for reasons that are described below. The pattern of results for CHL and LEABRA are largely as before, with some level of slowing due to the fact that only 1/5 of the total item by question permutations were presented in a given epoch. Given this relatively sparse level of sampling, it is clear that there was some transfer of the learning that took place on a given item to other instances of that item where different questions were asked. The generalization performance was essentially identical across the different algorithms, and not substantially different from generalization levels on other versions of the task. In sum, these results indicate that interactive networks can be trained to provide flexible access to encoded knowledge, and thus provide at least a starting point for thinking about how the same feat might be accomplished in humans. Further, they provide a concrete justification for the use of interactive networks, which have been shown to otherwise incur generalization penalties on other tasks.

Perhaps the most surprising result from this task was the complete failure to successfully train the interactive Almeida-Pineda (AP) backpropagation algorithm on this task. For none of a wide range of learning rates was any sign of learning progress evident over 1000 epochs of training (see Figure 7.16 for sample learning curves). These networks were identical to the AP networks that learned the standard uni-directional version of this task as described above, with the simple addition of full bidirectional connectivity with the input/output layers, and the disambiguating "sex" layers. The most likely reason for this failure is that the learning in one direction interfered with the learning in other directions, which is in contrast to the transfer evident with CHL and LEABRA. A significant difference between AP and both CHL and LEABRA is that these latter algorithms explicitly preserve the symmetry of the reciprocal weights, while AP does not. This lack of symmetry preservation in AP

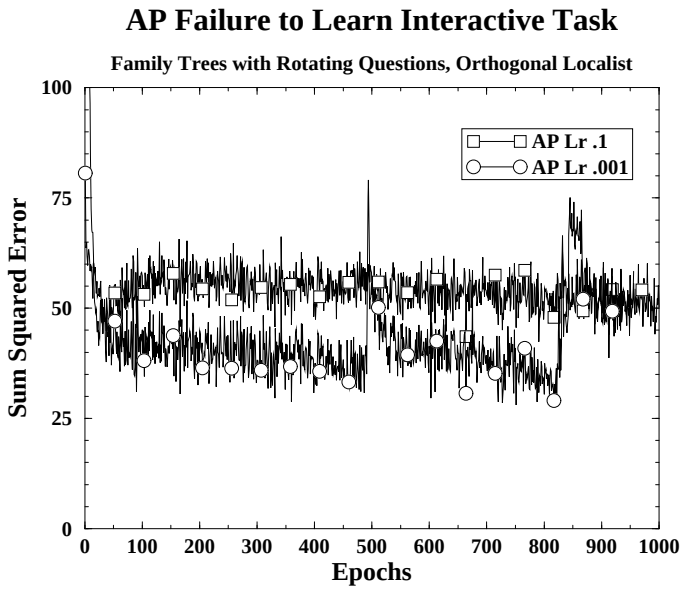

Figure 7.16: Learning curves for AP algorithm in an interactive version of the family trees task where the inputs and questions are rotated on each trial. Networks were unable to learn the task using a wide range of learning rates (.1, .001 shown, .02 also tried with similar results).

was seen to be an important correlate of its relatively better generalization performance compared to CHL, since it results in a less interactive network. The results on this task show that there is a cost associated with this lack of symmetry, which comes in the inability to learn to flexibly access knowledge in a task such as this one.

Comparison with Other Techniques

Finally, it should be noted that at least one other approach has been taken to speeding up learning in deep networks. Schraudolph and Sejnowski (1996) have developed a technique for setting the learning rates for standard feedforward backpropagation networks that results in faster learning on the family trees task. This technique, called *tempering*, sets the learning rate for each layer in the network so as to make the change in activation state that results from changing the weights proportional to the same constant throughout the network. Further, a "shunting" technique was used to allow learning of the bias weights to rapidly get rid of any constant bias in the error term. In the application of this idea to the family trees network, the scaling factors for the local learning rates were 1.5, .25, .1, and .05 respectively for layers increasingly far away from the output layer. The result was a substantial increase in the learning speed in this problem, from 2,438 epochs for batch mode learning with momentum to 142 epochs. Further, when the delta-bar-delta adaptive learning rate function was used in addition, networks learned in as fast as 61.7 epochs. This is still nearly twice as slow as the fastest LEABRA results (33 epochs), but they used the original 6/12 hidden unit configuration, so one might imagine that they could speed up learning further by using more hidden units. When 60 hidden units were used with simple on-line learning in BP, learning time was roughly equivalent

to their results (68 epochs). They did not present any generalization results. One general problem with this technique is that it would not work well in an interactive context like the one just described, since it requires that the learning rates be scaled as a function of the distance from the output. In the interactive problem, and presumably in the brain, the source of error signals can vary considerably, and the need to readjust the learning rates based on this variation might be problematic.

Chapter 8

Conclusions and Future Directions

The primary focus of the work presented in this thesis has been on two very basic and central aspects of neural network learning — generalization (both within and across tasks), and learning speed in deep networks. These issues have been investigated through a number of tasks and by comparisons among a range of different learning algorithms. The overall conclusions from this investigation are:

Interactivity can impair generalization, and learning in deep networks: The interactive CHL networks almost always generalized worse, and learned more slowly in deep networks than feedforward BP networks. The semi-interactive AP networks performed intermediate between BP and CHL. Direct examination of the sensitivity of the CHL networks to changes in input stimuli confirmed that this was a likely explanation for their poor generalization. Nevertheless, interactivity was essential for providing flexible access to knowledge in the family trees task, and is supported by a range of both psychological and biological data. Finally, AP networks were unable to use their interactivity to solve the interactive family trees task with rotating questions, indicating the importance of symmetry in the learning rule and weights for interactive networks.

Activity regulation and competition via the ReBel soft-kWTA function improves generalization, and learning in deep networks: Even without the Hebbian associative learning component, LEABRA networks with the ReBel activation function generalized better than their CHL counterparts. This can be attributed to the development of categorical representations (entropy reduction) and by damping the sensitivity of the interactive network. This provides an important functional role for the extensive inhibitory interneuron circuitry of the neocortex.

Hebbian associative learning via the MaxIn function improves generalization, and learning in deep networks: In most cases (with the exception of the purely combinatorial task studied in Chapter 6), the use of MaxIn in LEABRA resulted in better generalization, and faster learning in deep networks. This can be attributed to the development of useful representations according to the principles of entropy reduction and information preservation. This provides an important functional role for the associative LTP observed in the neocortex.

Error driven learning is essential for actually learning tasks: In no case was purely associative learning capable of learning a task to criterion, and in many cases performance remained quite bad throughout training. This reinforces the notion that the neocortex must be capable of performing error-driven learning. A biologically feasible mechanism for error-driven learning was presented in Chapter 2.

Auto-encoder (AE) LEABRA performed substantially better than all other algorithms: For both the digit recognition and the combinatorial domain tasks, the level of generalization exhibited by AE LEABRA could be of important practical interest, in addition to its potential application in understanding learning in the neocortex.

Adapting activity constraints did not perform nearly as well as ReBel: The fixed, *a priori* model provided by ReBel was found to be an important implementational advantage over purely adapting forms of activity constraints, which end up competing with and detracting from learning performance on the task.

LEABRA performed better than standard regularizers like weight decay and noise: Especially when the joint function of learning speed in deep networks and generalization capability is considered, the self-organizing learning in LEABRA represents a better way of constraining error-driven learning than these other methods.

Future Directions

While the relatively simple tasks used in this thesis enable the thorough comparative investigation of an algorithm's behavior, they provide only limited models of interesting psychological and neurobiological phenomena. Thus, future work with the LEABRA algorithm will be focused on the exploration of psychological and neurobiological phenomena for which LEABRA appears to be particularly relevant. A brief description and, in some cases, preliminary results, of these investigations are provided below.

It should also be noted that there might be practical applications of the LEABRA algorithm. As was mentioned above, the generalization performance of AE LEABRA is significantly better than other algorithms, and thus might find practical application. Further, there has been some interest in trying to extract rules from trained networks — this should be much easier to do in LEABRA than in a standard error-driven algorithm.

Psychological Phenomena

Multiple Constraint Satisfaction

An important use of recurrent or interactive networks is to solve multiple constraint satisfaction problems. In such problems, the consistency relationships among a set of properties or features are

expressed via the weights connecting units representing these features. A highly consistent configuration of such features can be extracted from a recurrent network by clamping some feature units, and letting the network settle according to the weight values. The stable network state reflects the satisfaction of multiple (possibly conflicting) constraints reflected in the clamped units and the weights. Multiple constraint satisfaction provides a potentially useful account of many aspects of human cognition, including the integration of multiple visual cues to determine properties of objects like depth, orientation, motion, etc. However, as the number of features is increased in such systems, it can begin to take an excessively long time for these networks to settle into meaningful states. This is due to increased difficulty in finding a local minimum in the high dimensionality of an activation space with many features. Thus, it is not clear if a generic recurrent network could settle fast enough to provide a useful model of multiple constraint satisfaction in visual processing.

The activity constraints of the ReBel activation function used in LEABRA provide a potential solution to this problem. As was argued in Chapter 4 and shown, if somewhat indirectly, in the simulations presented in this thesis, ReBel restricts the activation space searched through settling, and thus settles relatively rapidly and is less sensitive than a generic interactive network like CHL. For example, in many of the tasks (e.g., family trees, the combinatorial domain), CHL required as many as 160 cycles to settle, while LEABRA never required more than 50, and typically settles in around 20 cycles. Nevertheless, these tasks are not specifically multiple constraint satisfaction tasks.

An interesting test of the idea that LEABRA can provide better performance in multiple constraint satisfaction tasks is reported in a model of figure-ground organization (Vecera & O'Reilly, submitted). This model produces a representation of figural elements of simple line drawings by finding the most consistent interpretation of a configuration of otherwise ambiguous line elements. It uses factors such as concave regions defined by multiple line elements, and feedback from an emerging interpretation of the figural region, to settle on a figure representation. We used this model to account for familiarity effects in figure-ground organization by including object-level representations above the level of figural processing, with interactive connectivity so that emerging object-level representations will feed back and influence the figural representation. Thus, the entire network is quite large, having a 16x16 figural region, with four potential line segment/orientations per pixel, and performs multiple constraint satisfaction over several different levels of representations.

As a result of its size and complexity, a version of the model using standard CHL-like activation dynamics required 400 cycles of settling (including an elaborate annealing schedule), and even then produced an unacceptably large number of uninterpretable figural representations (as high as 44.5%). In contrast, the same model using the ReBel activation function used in LEABRA (note that there was no learning in this model) settled in 100 cycles, and had only 6% uninterpretable figural representations. Thus, the soft k-WTA activity constraints provided by ReBel caused the network to more rapidly and reliably find good solutions to this constraint satisfaction problem. This result should hold quite generally, and should be investigated further, in order to more fully understand the relationship between settling time, activity constraints, and performance.

Learning Spatially invariant Object Representations

In O'Reilly and Johnson (1994) and O'Reilly and McClelland (1992), we presented a model of how a neural network could learn to develop invariant object representations by capitalizing on a special training signal present naturally in the environment. This training signal arises because of the tendency for objects to persist in the environment, but yet to move around relative to our retinal reference frame (i.e., due to saccades or relative movement). Thus, we experience a sequence of images that all correspond to the same object(s) in different spatial locations, and this sequence provides a useful signal for the unsupervised learning of object representations, since we can effectively assume that over some time period, the retinal images should all correspond to the same high-level object representation. I showed that by building a hysteresis bias into the network, which maintained unit activity over time (e.g., through the effects of recurrent excitatory connectivity and lateral inhibition), units would develop invariant representations when presented sequences of images of an object in different locations. This model was based on the use of Hebbian associative learning, which would cause units to become associated with the different views of a given object, thus making them invariant.

All three of the important aspects of this model are present in LEABRA — interactive (recurrent) connectivity, lateral inhibition (as implemented by the ReBel function), and Hebbian associative learning. However, LEABRA adds a further component, error-driven learning, which could potentially play an important, but as yet unexplored, role in such a model. The problem with the existing model is that it often gets stuck in sub-optimal representations, where a single unit ends up with a representation of multiple objects in several locations, not a single object in multiple locations. This is due to the positive-feedback nature of associative learning, which leads to a rich-get-richer phenomenon, and the overextention of such representations.

This is exactly the kind of problem that error-driven learning can solve. For example, if a LEABRA network were presented translating sequences of objects, and error signals were available to shape the representations in addition to associative learning, the error signals could prevent the development of ineffective representations. These error signals could arise from a number of plausible sources: 1) reconstructing the current input, as in the auto-encoder version of LEABRA; 2) predicting the next input, which amounts to a modified version of AE LEABRA (more on this below); 3) using the invariant representations to label (identify) objects, and getting feedback about the accuracy of these labels from the environment. Preliminary research with 1) and 3) has shown that these error signals have the expected effect in the simple model used in O'Reilly and Johnson (1994). This model has "objects" which overlap by 1 out of 3 features, and this leads the purely associative network to develop multi-object representations in 3 out of 10 networks. In contrast, using LEABRA with error-driven and associative learning, no such representations developed. Future work on this topic will involve the use of more complex stimuli and will investigate the development of invariant, hierarchically organized feature, as well as object, representations.

Functional Models of Biological Systems: Hippocampal-Neocortical Interactions

The LEABRA framework has already proven itself useful in developing models of the interactions between the hippocampus, pre-frontal cortex (PFC), and posterior associational neocortex. A LEABRA model of these different brain regions has been developed, and data from normal humans and PFC-lesioned patients on an episodic memory task (Gershberg & Shimamura, 1995) successfully simulated. Critical to the success of this model was the ability to express the different characteristics of these brain regions by parameterizing the LEABRA model. For example, our model of the hippocampus (O'Reilly & McClelland, 1994; McClelland et al., 1995) relies critically on the sparse level of activity in areas DG, CA3, and CA1 of the hippocampus. This was easily and robustly implemented using the k-WTA activity constraints in LEABRA. The other areas had higher activity levels, but the k-WTA aspect was still important (especially in the case mentioned below). In addition, interactive processing was critical for this model, as all three brain areas had to mutually constrain each other's activity states. In this simple model, purely associative (MaxIn) learning was used, since the majority of learning that took place occurred in the hippocampus. It is interesting to note that the relevant areas of the hippocampus lack bidirectional connectivity, and thus are only capable of associative learning in the LEABRA model (since bidirectional connectivity is necessary for communicating error signals).

In a related model of these brain areas, which accounts for the role of the hippocampus (and to a lesser extent the PFC) in the conditioning phenomena of blocking and latent inhibition, activity constraints played a critical role in the functioning of the posterior cortical component of the model. This is because, in the blocking task, top-down activation from the hippocampus led to the activation of one set of representations in the posterior cortex, but, critically, this led to a suppression of another set of representations due to the activity competition. This suppression of activity is what led to the blocking effect in the model, as was predicted in a verbal account in Cohen and O'Reilly (1996).

These examples provide an indication of the promise that LEABRA holds for developing biologically- based models of cognition. LEABRA has a set of parameters and learning rules that, while potentially available piecemeal in other models, are well integrated and robustly implemented. In short, the LEABRA model provides a unique and useful level of modeling that captures more biological detail than existing abstract neural network models, but is still sufficiently abstract and computationally efficient to enable models of complex behavioral phenomena to be implemented and, importantly, understood.

Learning Inflectional Morphology

The investigation of neural network models of past-tense inflectional morphology, begun by Rumelhart and McClelland (1986), has played an important role in the debate over the sufficiency of neural network models for capturing systematic or rule-like behavior. However, despite 10 years of work, a satisfying model of the central phenomenon in this domain, the U-shaped curve of correct

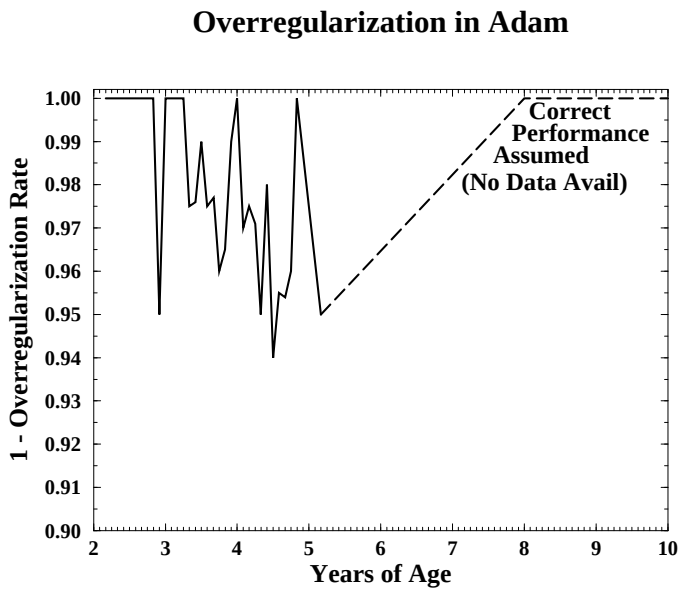

Figure 8.1: Overregularization rate (1 - overregularization errors) for Adam. Data reproduced from Marcus, Pinker, Ullman, Hollander, Rosen, and Xu (1992). Note that the data ends before the child attains perfect performance, so a projected line was drawn from the end of the data to the 8 year point, where it is assumed overregularization will have stopped. This makes clear the U-shaped nature of the curve.

marking of irregular words (see Figure 8.1 for an example), remains elusive. In simple form, the U-shaped curve consists of an early period where children correctly inflect irregular past-tense words, saying "went" for the past tense of "go". This is followed by a period where these words are overregularized, so that children say things like "goed" for the past tense of "go". Thus, they overzealously apply the "rule" which applies to the regular words to the irregulars. This phenomenon has been interpreted by some as an indication that people use symbolic-like rules in language production (Marcus, Pinker, Ullman, Hollander, Rosen, & Xu, 1992; Pinker & Prince, 1988). Finally, after some years of what amounts to only sporadic (and highly variable between subjects) overregularization of the irregulars, children learn to treat the regulars and irregulars correctly.

While there have been a number of models since the original work of Rumelhart and McClelland (Plunkett & Marchman, 1991, 1993; Daugherty, 1993; Hoeffner, 1996), and a considerable refinement of the characterization of the empirical phenomenon and the successes and failures of neural network models, the fact remains that no model has captured the U-shaped curve without reverting to implausible manipulations of the training environment or network training signal. Such manipulations amount to changing the frequency balance of regulars and irregulars, with a higher proportion of irregulars early in training (leading to correct irregular inflection), followed by the introduction of a large number of regulars. This leads to overregularization due to the large error signal introduced by the large number regulars. Similar manipulations have been performed by modifying the error signals themselves (Hoeffner, 1996). However, it is not likely that the child is actually exposed to such a changing linguistic environment (though such changes may be evident in their own production), making such manipulations implausible as models of the external environment.

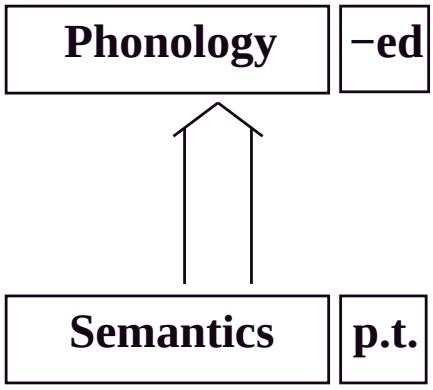

Figure 8.2: Schematic of the semantics to phonology mapping present in the past-tense inflection models used by Hoeffner (1996), and in the current simulations. The crucial point is that there is a strong correlation between the past-tense semantic (**p.t.**) and phonological features (**-ed**), and that associative learning will be sensitive to this correlation.

The problem with existing models can be traced to their reliance on purely error-driven learning. This is because the regular mapping, which is highly consistent and present in a large number of training items, will be learned relatively early. Thus, it will then cease to be a significant pressure on the error-driven learning process, as it will not generate substantial error signals. Accordingly, the irregular words are most likely to be overregularized *early* in training, and this tendency will decrease over training as the error pressure from the regular mapping decreases. This is consistent with the results from a large number of different error-driven neural network models (Hoeffner, 1996).

I propose that LEABRA will produce a U-shaped learning curve without any additional manipulations, as a direct result of the balance of error-driven and associative learning pressures in the model itself. The key idea is that associative learning will remain sensitive throughout learning to the strong correlations that exist between the semantic features for the past tense, and the regular phonological inflection ("-ed") that marks the past tense (see Figure 8.2). In particular, it is the case in LEABRA that early learning is dominated by error-driven learning (since the error signals are larger then), and after the error signals have been reduced, associative learning plays its biggest role. Thus, the shifting of the balance from error-driven to associative learning provides a mechanism for the transition from early correct performance to the period of overregularization. In the end, the network should be able to sort out the irregular and regular mappings, and achieve the final stage of correct performance.

This hypothesis was tested in a preliminary study using a LEABRA model of the semantics to phonology mapping task. There were 381 monosylabic English words with both a stem and a past-tense inflected version, each presented with the square-root of their actual frequency as given in Kucera and Francis (1967). The stem of the word was represented by a randomly selected pattern of 24 active units out of a field of 120 units. The inflectional semantics were represented by an additional 6 units, three of which were active for the stem form of the word, and three for the past-tense. The phonlogy of the word was represented with a vowel-centered, slot-based scheme, with three

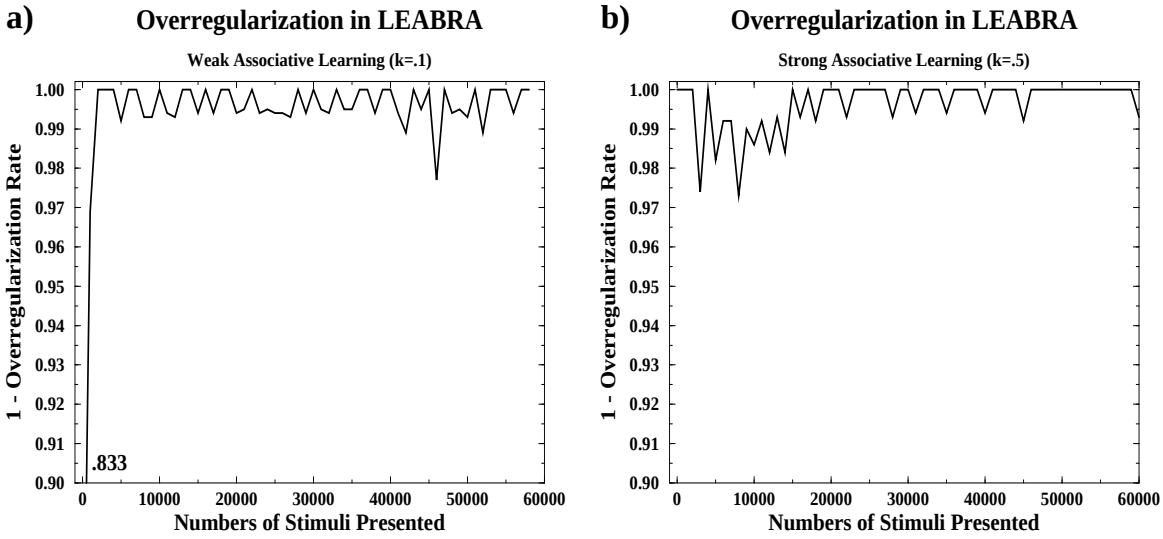

Figure 8.3: Overregularization curves from two versions of LEABRA that differ only in the strength of associative learning. **a)** shows a model with weak (.1) associative learning, and **b)** shows a model with strong (.5) associative learning. The stronger associative learning results in a U-shaped curve, while the weaker does not (note that the first point for the weak model is at .833, off the low end of the graph).

onset consonant slots, the vowel slot, followed by three additional consonant slots, the last two of which could contain the past tense inflection. For example, the word "talk" was represented as "tttOkkk" and its past tense, "talked" as "tttOktt". Each consonant was represented by a set of 18 units, 3 of which were active for any given consonant, representing commonly used phonological distinctions such as place and manner of articulation. Similarly, the vowels were represented by 17 units, 5 of which were active for any given vowel. These stimuli were adapted from models described in Hoeffner (1996). The network had 480 hidden units, with a 16.67% activity level in the hidden layer.

The critical test of the above hypothesis is whether associative learning is important for producing a U-shaped overregularization curve. Thus, two otherwise identical LEABRA models were run, one with weak and one with strong associative learning. Their overregularization curves are shown in Figure 8.3. The critical finding is that the model with strong associative learning exhibits a U-shaped overregularization curve, while the one with weak associative learning does not (note that the first point for the weak model is at .833, off the low end of the graph). Thus, given that the only difference between these models was the level of associative learning, and that purely error-driven models have not produced a U-shaped function with a static environment, this provides important evidence in support of the hypothesis that associative learning in LEABRA will lead to the U-shaped curve. However, there are many other aspects of the empirical data on the learning of inflectional morphology, and it is not clear if this preliminary model will provide a satisfactory account of all of it. Nevertheless, these results provide sufficient encouragement to pursue this important issue further.

There are a number of important consequences of the above preliminary result, which could become more solid when a more thorough modeling effort is undertaken. First, while it may be possible

to modify the environment or the error signals in a purely error-driven algorithm to obtain a fit to the empirical data, the LEABRA model has the potential to show this behavior as a direct consequence of properties that have been shown in this thesis to have a number of other important functional consequences. Thus, this would provide both a less *ad hoc* account of this data, and, at some level, the data could provide evidence that supports the LEABRA model of learning in the neocortex. Further, this result points to the importance of correlational structure for defining what is commonly described as "rule-like" behavior. It is quite possible that by building a sensitivity to this correlational structure into the learning algorithm (as is the case in LEABRA), many other aspects of human rule-like behavior can be modeled and understood.

Generalization in the Pronunciation of Written Words

Another language domain which has been the focus of considerable modeling efforts is the pronunciation of (monosylabic) written words (Seidenberg & McClelland, 1989; Plaut et al., 1996). The primary issues here are the extent to which a single system can pronounce both regular and exception words, the sensitivity of the system to the frequencies of items, and the ability of the system to generalize its knowledge to the pronunciation of nonwords. As was clear from the discussion of related issues in inflectional morphology above, and from the work on generalization presented in this thesis, it is likely that LEABRA's behavior on this task will differ in important ways from that of the standard error-driven algorithms that have been used to date.

In particular, the recent work of Plaut et al. (1996), hereafter referred to as PMSP, raises a number of important issues. First, as was noted in the introductory chapter, they used a backpropagation algorithm without a symmetry constraint to train an "attractor" version of their network, which, as has been shown in this thesis, does not develop significant reciprocal (bidirectional) weights. This lack of significant reciprocal weights in their model was confirmed by D. C. Plaut (personal communication). In addition, they encouraged the network to settle rapidly, which further decreased the extent to which their model could be described as a true attractor network. In the simulations reported in this thesis, the Almeida-Pineda (AP) algorithm performed intermediate between a feedforward and a fully interactive CHL attractor network on tests of generalization. It is likely that their model will perform even more like a feedforward network than the AP networks, due to the additional settling time pressure. Thus, their demonstration that an "attractor" network was capable of pronouncing nonwords (i.e., generalizing) to roughly the same level as a feedforward network (note that it performed slightly worse), is not surprising, and does not constitute a test of a true attractor network's ability to generalize in this task.

Given the results presented in this thesis, it is likely that the use of a true attractor network (e.g., CHL) will result in significantly worse generalization on this task. However, LEABRA should generalize at about the same level as a feedforward backpropagation network. This constitutes a set of predictions that should be tested. Unfortunately, the computational expense of running truly interactive networks on a large task such as this one (which has roughly 3,000 stimulus items, presented

according to a wide frequency range, in a network with hundreds units) has prevented any significant progress on this issue as of yet.

Another issue, which unfortunately requires even more computational power to address completely, has to do with the nature of the input/output patterns used in the PMSP model. These patterns were designed in such a way as to compress the regularities of the orthography-to-phonology mapping, using a number of convenient accidents of the structure of English spelling and pronunciation. These patterns should make it much easier for a network to pronounce nonwords, which is what they found compared to the much less systematic representation used in Seidenberg and McClelland (1989). However, it begs the question as to how such representations, which are presumably themselves developed in the neocortex somewhere, can arise. Further, it is apparent that these systematic input/output representations are quite different than the types of representations that develop in the hidden layer of the PMSP network, as the latter are highly distributed and share the relatively underconstrained appearance of the purely error-driven networks analyzed in this thesis.

The LEABRA algorithm could potentially address these issues in a couple of ways. First, the self-organizing learning in LEABRA has been shown to develop representations that more closely reflect the structure of the domain. Thus, it should be the case that a LEABRA model of the PMSP task will develop hidden unit representations that more clearly reflect the systematicities of the orthography-to-phonology mapping. Taking this one step further, it should be possible to implement a deep network version of this task, where the input and output patterns more closely resemble the surface features of the written and spoken words. Then, as in the family trees networks, internal coding layers could develop more systematic encodings of these surface features. One would expect the LEABRA version of such a deep network to learn more rapidly than the BP version, and, to the extent to which it developed more systematic hidden unit representations, generalize better. This would provide a much more satisfying model of human nonword pronunciation than the PMSP model, which accomplishes this task largely through the use of systematic input and output patterns.

Learning Temporally-Extended Behaviors

One important way in which a network becomes very deep involves learning over time. This is most clearly evident in the backpropagation-through-time algorithm, where a new layer of units is introduced for each time step over which the network must learn. Consistent with their substantial depth, it is typically the case that networks trained to recognize or perform temporally-extended behaviors require extremely long training times. For example, a very simple recurrent network trained in an environment generated by a simple finite-state-grammar required 60,000 sequence presentations to learn this grammar (Cleeremans, Servan-Schreiber, & McClelland, 1989). Given the rapid learning speed in deep networks evidenced by LEABRA in the family trees task, it is likely that it will be capable of learning temporally extended tasks more rapidly as well. In a preliminary test of this idea, the simple finite-state-grammar task used by Cleeremans et al. (1989) was run in a LEABRA network. Instead of using a copy of the hidden unit activities as the temporal context for predicting

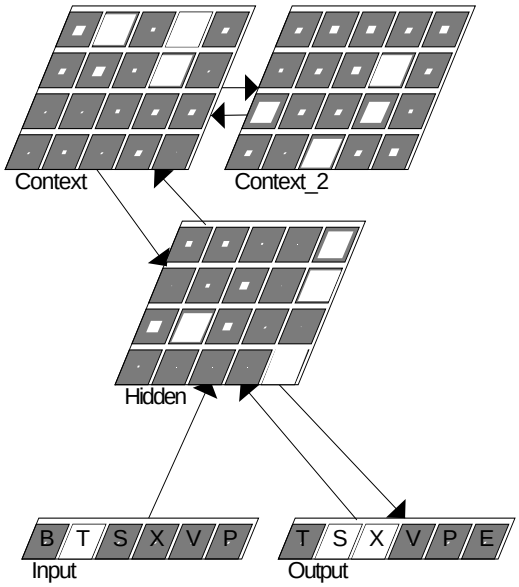

Figure 8.4: Network configuration used to learn a simple finite-state grammar in LEABRA. The two reciprocally connected context layers develop context representations that disambiguate the otherwise ambiguous input tokens according to the prior history of experience by the network.

the next symbol in the grammar, the LEABRA model used an additional set of context layers which were bidirectionally connected to the hidden layer, and to nothing else (see Figure 8.4). The network activities were decayed .9 of the way towards the average activation state after each training item, so that the prior activity of the network was partially preserved, serving as context for the processing of subsequent stimuli (just as in the auto-encoder version of LEABRA). The context layers were reciprocally interconnected to provide some stability of their representation over this decay. Finally, the connections in the network used a modified version of the LEABRA algorithm which was performed based on the receiving activations for time t, and the sending activations for time $t-1$, in addition to the standard learning on patterns at time t. This enabled the network to learn about temporal contingencies from one pattern to the next.

When trained in this manner on randomly generated samples from the finite state grammar, the network required only 3,690, SEM 1,260 sequence presentations to learn the problem. This represents a massive improvement over the results with the simple recurrent backpropagation network used by Cleeremans et al. (1989), and can be attributed to the self-organizing development of useful representations in LEABRA, compared to the slow accumulation of error-driven representations in the BP network. While this is promising, appropriate control experiments must be done to determine the effect of variables like hidden layer size, and the architectural differences between these networks. In any case, the successful use of the more "naturalistic" context layer for solving this type of task is of considerable interest, as it does not require unrealistic copying of unit activities, and instead depends on the development of useful representations in an otherwise unconstrained layer. It is likely that the success of this architecture relies on the use of the activity constraints and associative

learning in LEABRA, but this has not yet be systematically tested. Finally, it should be noted that the role of the context units in this task is similar to that proposed for the pre-frontal cortex (PFC) (Cohen & O'Reilly, 1996; Cohen & Servan-Schreiber, 1992; Cohen et al., 1990), and that the ability of LEABRA to develop useful representations in this type of layer could be useful for modeling this area.

Learning Multiple Output Patterns

It is often the case in psychological modeling that one needs to train a network to respond to a given input in different ways, often without any particular distinguishing feature to determine which way to respond on a given trial. In other words, it can be useful to learn one-to-many mappings. For example, there are many different words that express the same underlying concept, and it would be useful to have a network pick, essentially at random, one of a set of possible words in order to express this concept. The family trees problem, for another example, had some cases where there were multiple equally valid answers to a given question. If one is using orthogonal, localist representations, it is always possible to simply activate all of the possible outputs. However, this is not generally possible when using more plausible and powerful distributed representations. If the outputs are instead paired randomly with the input patterns on different training trials, backpropagation networks tend to produce a single "blended" output pattern that represents an average of the desired outputs.

One approach to this problem was suggested by Movellan and McClelland (1993), who derived a learning algorithm that specifically tries to match the probability distributions associated with multiple possible output patterns. However, this algorithm requires prohibitive amounts of computation in order to obtain adequate samples of the relevant statistics. I propose instead that LEABRA can be used to solve this problem in a much simpler way. The idea is that the activity constraints present in LEABRA, together with its interactive attractor dynamics, will cause the network to settle into specific trained attractor states corresponding to the different possible outputs, instead of producing a blended combination of these outputs as is typical of a feedforward backpropagation network.

This idea was tested using a simple three-layer network, with 25 units in each of the three layers. 13 training items were created with one input pattern and two possible output patterns per item. The patterns were random with 6 out of the 25 units active, and a maximum overlap of 2 bits with any other pattern. Both input/output pairings of each item were presented in each epoch, resulting in 26 training items per epoch. For each item during training, the output pattern was compared to all of the 26 different possible output patterns, and the one that was closest to this pattern (using a sum-squared distance measure, with a tolerance of .5, so that if all units were on the right side of .5 the distance would be 0) recorded. Two statistics were computed (the average of the best of each of which are reported below): 1) the percentage of training items for which the closest pattern was one of the two correct output patterns; 2) the percentage of training items for which the output activity was also on the right side of .5 for all units in this closest pattern. The first measure could be satisfied with an

appropriately blended representation of the two items, while the second requires that the network produce distinct outputs that match either one of the two trained outputs.

Three types of networks were compared: BP, CHL, and LEABRA. The BP network achieved 100% on the first measure, but 0% on the second one. Thus, as would be expected, it produced a blended output pattern that was close to the targets, but never actually matched either one of them. CHL achieved 96% on the first measure, and 7% on the second one. Thus, it was slightly worse at producing an output that was close to the targets, but very occasionally managed to settle into an output state that matched one of the targets. This is probably due to the attractor dynamics in this network. The results from LEABRA were dramatically better, achieving 100% on the first measure and 94% on the second one. Thus, it always settled into a close output pattern, and most of the time this pattern actually matched one of the targets. This can be attributed to the combination of attractor dynamics and activity constraints in LEABRA, so that it would settle into a trained configuration with the appropriate number of active units, and not a blend or other spurious attractor state. Finally, in a version of LEABRA without associative learning, the score on the second measure dropped to 91%, indicating that associative learning is playing some role in this phenomenon — it probably helps to solidify each of the output states as a coherent (self-correlated) attractor state.

Priming Effects

It is well known in the psychological literature that the mere exposure to a stimulus will lead to subsequently faster processing of that stimulus, and that this effect, known as repetition priming, is relatively long lasting (hours, days, even months). Attempts to use error-driven learning algorithms to model this effect have been problematic, since it is not obvious where an error signal of any magnitude would come from based on the mere exposure to a given item. However, it is clear that associative learning would produce the desired learning signal based on the simple activation of representations triggered by the presentation of an item. Thus, LEABRA provides a useful framework with which to model such effects. Indeed, Stark (1993) found that the associative learning in LEABRA was able to model priming data from human subjects, but error-driven learning was not.

A simple experiment with trained versions of the multiple-output networks described above serves to demonstrate the difference in LEABRA's priming behavior compared to CHL and BP. There is an obvious measure of priming in these networks — the extent to which the network produces the same output pattern it was last exposed to in conjunction with a given input pattern. As it happens, the items in the training set were arranged so that all items with the first output pattern appeared sequentially in the training set, followed by all items with the second output pattern. During training, the items were presented in permuted order. However, for this experiment, they were presented sequentially. Thus, if priming were taking place, one would expect that, after having been exposed to the first set of output patterns, the network would produce the first output pattern during training on the second set of output patterns. Similarly, training on the first set of output patterns at the start of the next pass through the stimuli should produce outputs in the second set of output pat-

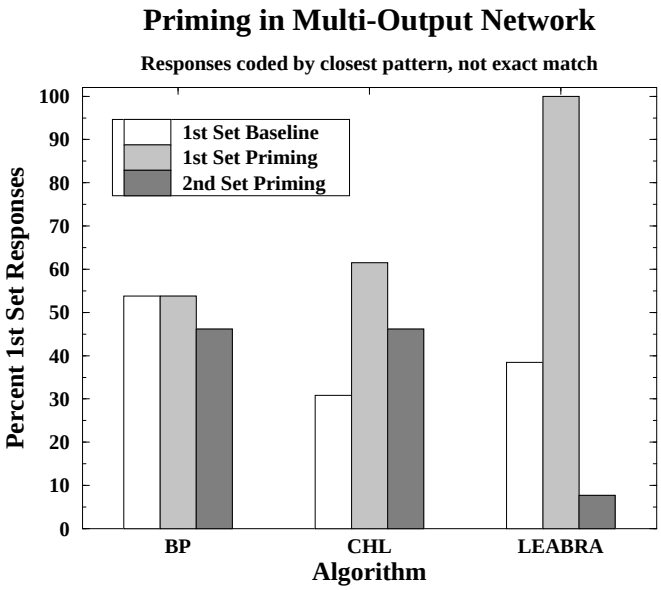

Figure 8.5: Priming effects in multiple-output network. **1st Set Baseline** is baseline performance for output patterns in the first set, prior to specific training. **1st Set Priming** is responses to first set after one training exposure to output patterns in the first set. **2nd Set Priming** is the reversal of this priming by exposure to the 2nd set of output patterns, which should result in fewer 1st set responses (and more 2nd set responses).

terns, as a result of training taking place at the end of the prior epoch. This test is particularly nice because it allows for intervening training on the other items before assessing priming effects, which simulates the delay (and potential interference) present in human priming experiments.

The data for this experiment, shown in Figure 8.5, support the hypothesis that LEABRA exhibits much greater priming effects than BP or CHL. The BP network showed little evidence of any priming at all. While the CHL network did show some effect of training on the first set of patterns, and a reversal of this from training on the second set, it is important to note that in none of these cases was the network actually producing the correct output pattern (to within .5 tolerance). Instead, it was merely shifting the response so that the output was closer to one set than another. The difference between the BP and CHL networks is probably due to the attractor dynamics in the CHL network, which allows small weight changes to have some effect on output state. In contrast with CHL, the LEABRA network was actually producing the correct output pattern, and, as is evident in the figure, is capable of shifting which of the two output patterns is produced based on one training trial. Thus, the LEABRA network clearly has two well-defined attractor states for each input pattern, and shifts easily between the two based on mere exposure. Both activity regulation, which encourages the network to settle into existing attractors, and associative learning, which provides a larger learning signal from mere exposure to the items, are important for LEABRA's performance in this example. Future work on this topic could make closer contact with detailed patterns of data from human priming studies.

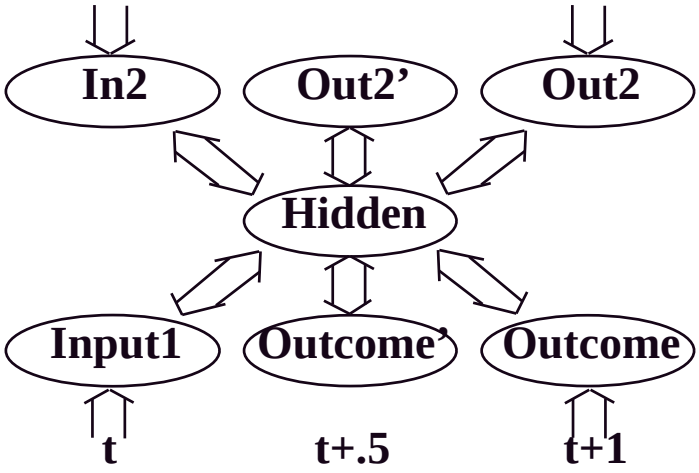

Figure 8.6: Possible mapping of expectation-outcome phases onto neural system where outcome must be experienced through same channel as original inputs. Primed notation indicates the production of an expectation based on hidden unit activity, while non-primed are experienced states from the environment. Note that there are only 3 layers, but the different states of the input/output layers are splayed out over time. Learning may also come from interactions between different input pathways (modalities, streams, etc), where one pathway serves as the "output" for the other.

Neurobiological Phenomena

In addition to the above topics of future research on the functional and psychological uses of LEABRA, there are a set of important issues regarding the neurobiological status of the algorithm that require further investigation. These are briefly described below.

The Nature of GeneRec Phases in a Sensory System

One of the important properties of the GeneRec algorithm is that it allows for simple activity states to provide error signals. This was explained in Chapter 2 in terms of the generation of expectations, followed by the experience of the actual outcomes. While this makes intuitive sense, and can easily be mapped onto existing neural network architectures, there are some important unresolved issues regarding its mapping onto processing areas in the neocortex. The central problem is that, unlike in a standard three-layer network, the experience of an actual outcome is presumably occurring via the very same sensory inputs that gave rise to the generation of the expectation. Thus, it is not particularly plausible to imagine that the original inputs are still present over an "input" layer, and that the outcome gets mapped magically onto a separate "output" layer.

One possible mapping of expectation and outcome signals that avoids some of these problems is shown in Figure 8.6. This configuration is similar to the auto-encoder LEABRA model, except that instead of reconstructing the input over the sensory input layer, the network predicts the next outcome over this input layer. Then, when the next outcome is actually experienced, the difference in activation states between expectation and outcome can serve as an error signal. The outcome

at time $t + 1$ would then serve as the "input" for the next time step, from which another expectation would be generated, and so on. Note that if the expectation about the subsequent outcome were that it would remain the same as the current input, this would be roughly equivalent to auto-encoder LEABRA. There is some indirect evidence that something like this could be happening in the neocortex. Duhamel, Colby, and Goldberg (1992) found that the anticipated perceptual effects of a saccade are seen in the parietal cortex prior to the actual experience of the results of the saccade. Thus, it could be that this reflects the generation of an expectation prior to experience of an actual outcome, as would be required by the proposed mechanism.

Also shown in the figure are inputs from another processing pathway, which could be another modality, or just another "stream" of processing of the same modality. Such inputs, converging on the same hidden layer, could play the role of the "output" units in a more standard three layer error-driven network. In this case, input from one pathway participates in the generation of an expectation for the outcome in another pathway (e.g., you see two cars colliding, and expect to hear a loud sound). Then, an outcome (the sound) is experienced in this other other pathway, and the difference in expectation and outcome states in this other pathway will train the hidden units to better predict the relationship between these two pathways.

This type of system for error-driven learning is obviously more complicated than the simple form used in the simulations in this thesis. It may place additional constraints on the activation and learning properties of units in the network, and the way in which their sensitivity to constant input activity is modulated (e.g., by habitation or by modulatory neurotransmitters like dopamine and norepinephrine). For example, the world presents continuous input to the retina, and yet this scheme requires that the visual neocortex should alternately be representing an expectation of a future outcome instead of being driven entirely by the retinal input. Finally, it is also likely that these modulatory neurotransmitters, which have been shown to have phasic firing associated with salient environmental events (Schultz et al., 1993; Montague, Dayan, & Sejnowski, 1996), could provide a "learn now" signal that synchronizes learning around a salient outcome. Chapter 2 contains a related discussion.

Details of Activity Regulation

As was discussed in Chapter 4, the ReBel activation function is intended to capture the effects of inhibitory interneurons on excitatory pyramidal cells in the neocortex. However, the ReBel function itself is rather abstract, and does not directly implement inhibition as such. Thus, an important next step is to develop more detailed models of this inhibitory and excitatory circuitry, and attempt to relate its properties to those of the ReBel function. This would probably involve the creation of point-neuron models with discrete spiking, and the modeling of a relatively large number of units in a single patch of neocortex, in order to average out the noise present in such systems. Such a model would probably be too complex to simulate behavioral data, but it could serve as a bridge, through the more abstract LEABRA model, between behavioral models that depend on activity regulation and the biological substrates that actually implement it.

Details of Synaptic Modification

A synaptic modification mechanism that would be capable of performing error-driven learning in the neocortex was described in Chapter 2. This mechanism could be implemented in a biophysical model of a neuron, and its detailed response to inputs and postsynaptic activity of different magnitudes and timing studied. This would provide a means for generating critical predictions that could then be tested in LTP/LTD studies in the neocortex. Further, such a mechanism could be combined with the more detailed model of excitatory and inhibitory neurons in the neocortex to provide an implementation of LEABRA at a more detailed biological level. One can imagine that this more detailed model might have important functional differences compared to the more abstract LEABRA model used in this thesis. In any case, if the LEABRA model is to be taken seriously as a model of learning and processing in the neocortex, this next step will need to be taken.

Chapter 9

Appendix: Implementational Details of LEABRA

Every learning algorithm has two faces: the theoretical and the implementational. The theoretical level describes what computational objectives the algorithm is trying to achieve, and generally how it goes about doing so. LEABRA has been described at this level in the main body of the thesis. The implementational level contains a number of details, some of which can be important for the algorithm's performance, which somehow fall below the threshold of pure *a priori* principles (e.g., the use of "momentum" in backpropagation). Most of the details associated with LEABRA are devoted to making the ReBel activation function robust across arbitrary network configurations. Because ReBel is a fixed (non-adapting) function, the normal process of learning does not necessarily result in optimal activation parameters for a given problem. Thus, additional adaptive mechanisms have been added. While these could in principle be implemented manually, it is much simpler on the user to incorporate them in the model. Thus, much of the implementational detail in LEABRA can be seen as a consequence of the use of fixed constraints, and of the increased complexity associated with making the mechanisms robust, as discussed in Chapter 3. Hopefully, the advantages of using such fixed constraints and robust mechanisms as demonstrated in the thesis outweighs the cost of this additional detail.

It is important to emphasize that the parameters associated with the following implementational details are never changed from one simulation to the next (except in cases where their importance is being tested), and thus do not constitute additional free parameters for any given simulation. Further, the default parameter values are all very robust and were set by a coarse parameter search along roughly logarithmic values (1,2,5, etc). This reflects the use of the implementational principle favoring robustness. The contribution to the performance of LEABRA of any given item does not make or break the algorithm. Each one simply improves things a little bit, but the aggregate probably performs significantly better than a version of LEABRA without them. Finally, it should be noted that most of these details are necessary only because of the particular soft kWTA formalism (ReBel) used

in LEABRA, and that a more biologically realistic implementation would use actual inhibitory interneurons and presumably have its own ways of dealing with the issues covered by these details.

The main body of the details are divided into two sections, one covering the learning rule, and the other the activation function. Following this, a pseudo-code implementation of LEABRA is provided, which grounds everything in a concrete instantiation that can make the algorithm easier to understand. Next, a section on the derivation of the GeneRec learning rule for the GausSig individual probability function is presented. Finally, there is a discussion of some general heuristics or guidelines for setting parameters in LEABRA, mostly having to do with the size of the network and the level of activation in the layers. These architectural parameters affect the variance of net input experienced by different units, which must be compensated for in different cases by changing the level of competition or balancing the contributions from different layers.

Learning Details

Combining Error Signals and Associative Learning

The MaxIn associative learning can be combined with the error-driven learning of GeneRec in a simple additive manner. However, adding an exception for the case when the error signals indicate the weight should be decreased, and the associative learning indicates that it should be increased, seems to work better in general. In this case, the error signals take precedence, and no associative learning is done. This enables the error signals to eliminate "bad" representations, which would otherwise be reinforced by the associative learning.

Soft Weight Bounding

The weights in LEABRA are thought of as representing something like the conditional probability of an input value given that the receiving unit is active: $P(x_i|h_j)$. This is similar to the interpretation of the weights in competitive learning (Rumelhart & Zipser, 1986; Nowlan, 1990). As such, the weights should be bounded between 0 and 1. This can be done simply by clipping the weight values within this range, or it can be done by applying a *soft weight bounding* function which has the additional benefit of causing (over time averaging) the weight to equilibriate at a point around the conditional probability. The soft weight bounding function is applied to the delta-weight function computed by the learning rule (Δw_{ij}):

$$\Delta_{bounded} w_{ij} = \begin{cases} \Delta w_{ij}(1 - w_{ij}) & \text{if } \Delta w_{ij} > 0 \\ \Delta w_{ij}(w_{ij}) & \text{if } \Delta w_{ij} < 0 \end{cases} \tag{9.1}$$

The conditional probability properties of this function can be observed by considering the case that the weights are increased p proportion of the time and decreased the remainder, $1 - p$, and that the magnitude of the increase and decrease (denoted simply Δ) is the same. Thus, at equilibrium, the

expected weight value where the weight change will be zero under the above function is:

$$\begin{aligned} \Delta^{\infty} w_{ij} &= p\Delta(1 - w_{ij}) - (1 - p)\Delta w_{ij} \\ 0 &= p\Delta - w_{ij} p\Delta + w_{ij} p\Delta - \Delta w_{ij} \\ w_{ij} &= p \end{aligned} \tag{9.2}$$

Thus, the weight comes to equal p, which, assuming the weights are only adjusted when the postsynaptic unit is active, is $P(x_i|h_j)$. To the extent that the weight changes have different values, this will bias the probability measure in a corresponding manner. In general, this weight bounding function causes the weights to equilibriate at a point which reflects the balance of increasing and decreasing pressures on the weight. This weight bounding is used in all simulations. Test simulations without it reveal that it is not essential for learning many problems, but does result in faster and more reliable learning than without it. It also has a natural biological interpretation in terms of an adaptive process that is a function of the current level of the weight parameter. Thus a given weight change modifies the existing efficacy in percentages of its current value, instead of raw values.

Weighting Terms in MaxIn

The MaxIn learning rule has two components, which correspond to entropy reduction and information preservation. These also correspond to subtractive or zero-sum-Hebbian (ZSH) and multiplicative or soft-competitive learning (SCL) weight normalization, respectively. The ZSH term is gated by a measure of the extent to which the unit already has a high signal-to-noise ratio (SNR) (i.e., has already reduced the entropy). This term is essentially a complement of a likelihood ratio term, so it is denoted L_c:

$$L_c = 1 - \frac{1}{1 + \left(\frac{P(h_j|\mathbf{x}_p)}{P(h_j^{\nu}|\mathbf{x}_p)}\right)^{-\gamma}} \tag{9.3}$$

Since the MaxIn learning occurs on the plus-phase activations, this gating term is computed on these activations as well. Note that in the case that the activation of a unit has been clamped by external inputs, then a fixed L_c value is used, typically .4, which corresponds to a relatively high value of the likelihood function (.6 as compared to the "noise" input). Also note that the value of $P(h_j^{\nu}|\mathbf{x}_p)$, instead of being computed de-novo based on the actual definition used for the MaxIn rule, can be approximated by $P(h_q^r|\mathbf{x}_p)$, which already available from the activation function. This is done purely as an implementational optimization, and it does not appear to affect the efficacy of the algorithm.

Also, the strength of the SCL term of MaxIn can be parameterized by introducing a k_{scl} parameter which multiplies this term in the MaxIn function (4.41), and the weighting of the ZSH term by γ

removed:

$$\frac{1}{\epsilon}\Delta w_{ij} = y_j \left[\left(1 - \frac{1}{1 + \left(\frac{P(\mathbf{x}_p|h_j)}{P(\mathbf{x}_p|h_j^p)} \right)^{-\gamma}} \right) (x_i - \mu(\mathbf{x}_p)) + k_{scl}(x_i - w_{ij}) \right] \tag{9.4}$$

Assuming a γ value of 2, which is typically used, a k_{scl} value of .5 will bring these terms into the proper relative strengths. However, it is useful to be able to manipulate the strength of this term to determine the relative contributions of ZSH and SCL to learning in LEABRA.

Activation Function Detials

Updating Prior Probabilities Over Settling

The ReBel activation function (4.21) has prior probability terms which can be iteratively updated to implement a settling process in an interactive (recurrent) network. This amounts to simply setting the prior probability of a unit to its probability (activation) on the previous time step. The prior is updated after every cycle of processing until the difference between the current probability and the prior goes below some threshold (.02 is standard). Most networks will require a total of no more that 60 cycles, and typically around 20 to 30, to reach this criterion.

The formal legitimacy of this settling procedure is somewhat questionable, given that the prior probabilities are technically supposed to be updated only after receiving *independently distributed* data drawn from some underlying distribution. However, in this case, the data from one cycle to the next is not a random sample from some underlying distribution, but rather is an evolving entity with correlations over cycles of settling. Nevertheless, the procedure converges well and produces interpretable activations which are obviously useful in guiding learning. More study needs to be devoted to the formal issues in this area, however.

Preserving Dynamic Range of Activations

This set of implementational details deal with the problem of keeping the independent probability values ($P(h_j^i|\mathbf{x})$) of the units within a useful dynamic range (i.e., preventing "floor" and "ceiling" effects). This is important for enabling comparisons amongst the units to be made via the log odds function used in ReBel. It is accomplished by setting the offset and gain parameters of the sigmoid that determines $P(h_j^i|\mathbf{x})$ as a function of the average and maximum net input in a layer. Thus, as the average net input floats around due to changes in input patterns and overall changes in weight values due to learning, the unit probabilities are automatically adjusted to compensate for these global changes. It is likely that the brain employs similar regulatory mechanisms through recurrent inhibition and neuromodulators.

The offset is set according to the following function:

$$\Theta = avg + k_{off}(max - avg) \tag{9.5}$$

where k_{off} is a parameter that determines how high to set the offset. This is typically set to .25. The offset is limited by a lower bound parameter, so if all units are receiving below this level of net input, none of them are particularly active. This enables the network to learn to inactivate an entire layer of units, if necessary.

The gain is set according to the following function:

$$\gamma^i = 2.9444/(max - \Theta) \tag{9.6}$$

which makes the maximum individual probability value in the network equivalent to the sigmoid of 2.9444, which is .95. This keeps the remainder of the values in the sensitive range of the sigmoid. However, early in settling, the units would end up with large gain values if this were applied at that point, since the maximum net input is not very different from the average. Thus, the floating gain is computed after some small initial number of cycles have been processed (typically 10). However, this results in settling on the initial cycles of a given pattern being dependent on the gain value of the last cycle of the previous pattern. Thus, the actual floating gain value is computed as a running-average of (9.6) with a small update rate (typically .001). This corresponds to something like a slowly adapting arousal modulation in biological terms.

Preserving Small Probabilities for Learning

The odds-ratio formalism at the heart of ReBel tends to make sharp distinctions between those units which are above the *kWTA* threshold, and those that are below it. This is necessary for the associative learning to perform correctly, as it requires distinctions in activity to perform the associative credit assignment problem effectively (i.e., those units which are active are associated with the current inputs, those which are not are not). However, it is also desirable to perform some degree of learning in those units which were below threshold, but which still received some amount of support from the current input (i.e., their independent probability was significantly above 0). These "fringe" units can be shaped and drawn in or away from representing the current input pattern, thus providing the benefits of a soft-competitive learning function as described in Nowlan (1990). This is especially important early on in learning when the unit's weights bear little resemblance to "well-formed" hypotheses about the domain. At this point, one does not want to over-commit to any given hypothesis, and the activities should be more graded in nature. Later on in learning, as the units shape their weights, the distinctions between units will become larger, and the competition will be sharper as a result.

For the above reasons, the activation of a unit in LEABRA is set to be a weighted sum of both

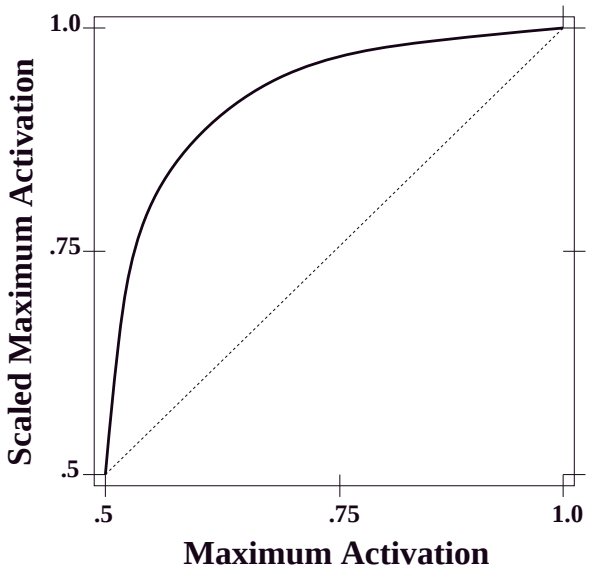

Figure 9.1: Activation scaling as a function of the maximum activation value in a layer. All unit activations are multiplied by the ratio of the scaled maximum value to the original maximum value.

the independent probability and the ReBel probability term:

$$y_j = \gamma_{raw} \; P(h^i_j|\mathbf{x}_p) + (1 - \gamma_{raw}) ReBel(h_j, \mathbf{x}_p) \tag{9.7}$$

The γ_{raw} or "raw gain" parameter determines the proportion of the activation which is determined by the raw independent probability. This parameter is typically .1, meaning that the *kWTA* constraint is still fairly strongly enforced.

Activation Scaling

As activation propagates through the network, there is some degree of loss through successive layers since hidden units have greater uncertainty (especially early on in training) and correspondingly lower activation levels than the inputs which feed into them. This cascades through the subsequent layers in the network, resulting in increasingly weak and undifferentiated activation states. One way of compensating for this loss is to increase the gain of both the individual probability sigmoids and the ReBel odds ratio sigmoid, but this sacrifices the graded activation signal which is crucial for learning as discussed above.

In order to compensate for the loss without sacrificing the graded signal, all the activations in a given layer can simply be scaled up by a single multiplicative factor. This factor is a function of the maximum activation in the layer. However, it is not desirable to have the maximum activation of every layer be identical, so the scaling is not a linear function of the distance between the maximum and some target maximum. Instead, a convex-shaped function of the current maximum is used (see Figure 9.1), so that the activation value is increased relative to its current value, but not to a fixed

constant value. This convex function is actually implemented by the top half of a sigmoid: $a_{scaled} = \sigma_{scale}(a_{max} - .5)$ with a gain of 10, so it is fairly highly curved over the .5 to 1 range of the maximum activation value. Every unit in the layer is multiplied by the ratio of a_{scaled}/a_{max}.

Thresholds

There are several thresholds that can be used to optimize the computational speed of processing in LEABRA. These are functionally relevant in some cases as well.

There are a set of thresholds that determine when to stop settling. Settling stops whenever the maximum change in activation (delta-activation) value for any unit falls below a threshold (typically .02), which typically results in a number of settling cycles between 15 and 40. A maximum cutoff of 60 cycles is used in case the delta-activation threshold was not reached.

Due to the $kWTA$ ReBel function, the units in LEABRA tend to either be active ($> .5$) or inactive. Processing can be speeded up considerably by using a threshold below which units do not send activity to other units (typically .1). To take advantage of this threshold, net inputs are updated by the sending unit, not the receiver. Thus, typically only around k units are sending activity in a given layer.

Similar to the sending threshold, a learning threshold can be applied. Since all weight changes are a function of the activity of the receiving unit, units with lower than a threshold level of activity (typically .1) in both phases do not perform any learning. This saves a similar amount of computation as the sending threshold, scaling the overall computational demands of LEABRA by k and not N.

Clamping Activation Values

There are some issues associated with the external "clamping" of unit activations given that such units also need to participate in learning, which in LEABRA depends on other activation-based parameters such as the L_c term described above and the gradient of the activations in the plus and minus phases. One alternative is to use "soft clamping", where the external input is just added into the net input for the units. A faster alternative is to clamp the individual probabilty value $P(h^i_j|\mathbf{x}_p)$ (to either 1 or 0, typically), and then compute the rest of the ReBel activation function as usual, using a $P(h^r_q|\mathbf{x}_p)$ kWTA threshold value equal to the parameter q (which is appropriate if there are exactly k values at 1.0, and the remainder at 0).

Given that the minus phase activations for the output units (those that receive plus-phase inputs from the environment) are typically lower than the clamped plus phase values, there exists a small positive error gradient (plus phase - minus phase) even when all the correct target values are among the k winners of the kWTA activation function. This small gradient is useful because it allows associative learning to occur (it does not occur if the error gradient is negative, as described previously), and tends to communicate a positive gradient to those units which were responsible for getting the right output units active in the first place. A similar kind of phase gradient can be implemented for the

input units, which facilitates learning over the input patterns just as the natural gradient does for the output units. This is done by moving the clamped activations in the minus phase towards .5, while leaving the plus phase activations alone, resulting in a positive gradient. The magnitude of this gradient is normalized across different activity levels by making it equivalent to that which would be exhibited by a network with 25% activity, and so that the net gradient over all the input units is zero:

$$a_j = \begin{cases} a_j - k_{grad}/\alpha & \text{if } a_j > .5 \\ a_j + k_{grad}/(1-\alpha) & \text{if } a_j < .5 \end{cases} \tag{9.8}$$

The parameter k_{grad}, which is how much the activation for a unit in a layer with 25% activity would differ in the two phases, is typically .005. Thus, it is a very small effect, but it results in reliable improvments in performance.

Bias Weights

Bias weights, or adaptive constants added to the net input of units, are essential for effective learning in LEABRA. Biologically speaking, these might correspond to variations in the overall excitability of a neuron, which could be regulated in a number of different ways. Computationally, bias weights enable units to have different thresholds of input to become active, which is equivalent to a prior probability. Unlike other weights, bias weights in LEABRA are not bounded. They are simply adaptive thresholds that are added into the net input of a unit. These weights do not enter into the average net input computation for the layer which is used to set the floating offset as described above.

Bias weights are adapted according to the error derivative with respect to the unit, or simply:

$$\Delta w_b = a_j^+ - a_j^- \tag{9.9}$$

Thus, bias weights do not have an associative function (since they are not associated with any useful stimulus). One problem with (9.9) is that the weights tend to grow large over time due to the accumulation of small activation differences. As described above, clamped output units are typically more active than in the minus phase, and this can also be true of hidden units in the plus and minus phase. Thus, a learning threshold (typically .1) is applied to the bias weights, so that any change below this threshold is not applied to the weights.

Weight Gain

Since all weights in LEABRA have the same range (0 to 1), it is not possible for the network to develop large weights to some inputs that are important but would otherwise contribute little to the overall net input of a unit. For this reason, a "weight gain" parameter is applied to all weights on a layer-by-layer basis in order to compensate for the case where a given layer has only one or a few active units, while other layers have many active units. This parameter simply multiplies the

weight values for the net input computation. If the difference in number of active units is small, it is not important to change this parameter from the default of 1. However, if a layer receives from one layer having 12 active units and another having only 1, for example, a weight gain parameter of 12 will equalize their contribution to the net input of the unit.

Pseudo-code For Implementing LEABRA

One effective way of understanding what actually happens in a given neural network algorithm is to view the code that implements that algorithm (an even better way is to write the code yourself!). The following pseudo-code shows how LEABRA is implemented. An actual implementation of LEABRA is available as an extension of the PDP++ simulator package, and is available by writing to the author (ro2m@crab.psy.cmu.edu).

This code is written for updating the activations and weights of units in a single layer (avoiding the looping over layers), and does not include details relevant for clamping external inputs, etc. An object-oriented coding style (like C++) is used.

```
// the following are the class objects and functions

// defines control of processing
class Control {
  // parameters
  int   n_cycles = 60;       // number of cycles for settling (max)
  float max_da_thresh = .02; // threshold for stopping settling

  // variables
  int   phase;               // current phase (0 = minus, 1 = plus)
  int   cycle;               // current cycle number
  float max_da;              // maximum delta-activation

  // functions
  Run() {                    // process one stimulus using LEABRA
    Do_Phases();
    Do_Learning();
  };

  Do_Phases() {              // iterate over plus-minus phases
    for(phase = 0; phase < 2; phase++)
      Do_Settling();
  };

  Do_Settling() {
    Layer.Initialize();
    max_da = 1;
    for(cycle = 0; (cycle < n_cycles) && (max_da > max_da_thresh);
        cycle++)
    {
      Layer.Compute_Net_Input();
```

```
      Layer.Compute_Act_i();    // individual probability
      Layer.Compute_Q_Val();    // comparison h_q value
      Layer.Compute_Act();      // final activity using rebel
    }
  };

  Do_Learning() {
    Layer.Compute_dWt();
    Layer.Update_Weights();
  };
};

// defines the variables in a layer
class Layer {
  // parameters
  int   n_active = 25% of units; // number of active units (k)
  float q_point = .25;       // where to put h_q between k and k+1
  float rel_gain = 2.0;      // gain of relative normalizing (nor_r) sigmoid
  float raw_gain = .1;       // amount of ‘‘raw’’ act_i in act
  float net_off_point = .25; // where to set net input offset

  // variables
  Unit  units[];             // an array of units for this layer
  float net_avg;             // average net input
  float net_max;             // maximum net input
  float net_off;             // floating offset for net input
  float net_gain;            // floating gain parameter based on net
  float kth_act_ip;          // kth most active (indiv*pri act) value
  float k1th_act_ip;         // k+1th most active (indiv*pri act) value
  float kth_act_i;           // kth most active (indiv act) value
  float k1th_act_i;          // k+1th most active (indiv act) value
  float q_val_ip;            // comparison h_q (indiv*pri act)
  float q_val_i;             // comparison h_q (indiv act)
  float act_max;             // maximum activation in layer
  float act_scale;           // scaling of activations
  float act_avg;             // average activation over layer

  // functions
  // initialization (before settling)
  Initialize() {
    for Unit* u over units {
      u->act = .25;          // reset activation, prior to mean
      u->p_act = .25;        // assuming 25% activity
      u->prv_net = 0;        // reset previous net input value
    }
  };

  // activation updating (compute in this order)
  Compute_Net_Input() {
    net_avg = 0;
    net_max = -MAXFLOAT;
    for Unit* u over units {
      u->Compute_Net_Input();// sets u->net as netinput
      net_avg += u->net;
```

```
    net_max = MAX(net_max, u->net);
  }
  // this implements the floating offset and gain for sigmoid
  net_avg /= units.size;   // divide by number of units
  net_off = net_avg + net_off_point * (net_max - net_avg);
  if(Control.cycle >= 10) {
    new_net_gain = 2.9444 / (net_max - net_off);
    net_gain = .001 * new_net_gain + .999 * net_gain;
  }
};

Compute_Act_i() {          // individual probability
  for Unit* u over units {
    u->act_i = 1 / (1 + exp(-net_gain * (u->net - net_off)));
    u->act_ip = u->p_act * u->act_i;
  }
};

Compute_Q_Val() {           // comparison distribution
  Sort(units by act_ip);    // actual sorting is optimized
  kth_act_ip = units[n_active]->act_ip;
  k1th_act_ip = units[n_active + 1]->act_ip;
  q_val_ip = k1th_act_ip + q_point * (kth_act_ip - k1th_act_ip);
  // these are required for computing nor_c for maxin learning
  kth_act_i = units[n_active]->act_i;
  k1th_act_i = units[n_active + 1]->act_i;
  q_val_i = k1th_act_i + q_point * (kth_act_i - k1th_act_i);
};

Compute_Act() {            // activation using rebel
  act_max = 0;
  for Unit* u over units {
    u->nor_r = 1 / (1 + (u->act_ip / q_val_ip)^-rel_gain);
    // preserve small probabilities for learning using raw_gain
    u->act = raw_gain * u->act_i +
                ((1 - raw_gain) * u->act_i * u->nor_r);
    act_max = MAX(act_max, u->act);
  }
  act_scale = 1.0;
  if(act_max > .5)          // activation scaling by max
    act_scale = (1 / (1 + exp(-10 * (act_max - .5)))) / act_max;
  act_avg = 0;
  Control.max_da = 0;       // control settling based on max da
  for Unit* u over units {
    u->act *= act_scale;    // actually scale activations
    act_avg += u->act;
    u->da = u->act - u->p_act; // change in activation
    Control.max_da = MAX(Control.max_da, u->da);  // for settling
    u->p_act = u->act;         // update prior from current
    // following only need be done at end of settling
    if(Control.phase == 0)
      u->act_m = u->act;       // store phase-based variables
    else {
      u->act_p = u->act;       // for use in learning
```

```
        u->nor_c = 1 - (1 / (1 + (u->act_i / q_val_i)^-rel_gain));
      }
    }
    act_avg /= units.size;       // also used in learning
  };

  // learning
  Compute_dWt() {
    for Unit* ru over units {
      for Connection* c over u.cons
        c.Compute_dWt(ru);       // call function on conns
    }
  };

  Update_Weights() {
    for Unit* ru over units {
      for Connection* c over u.cons
        c.Update_Weights();       // call function on conns
    }
  };
};

// defines the variables in a unit
class Unit {
  // parameters
  bool  GAUSSIG = true;      // true for GausSig (else Sigmoid)
  float net_dt = .7;         // dt for updating net input

  // variables
  float prv_net;             // previous net input
  float net;                 // net input
  float act_i;               // individual probability (P(h^i | x))
  float act_ip;              // individual times relative prior (act_i * p_act)
  float nor_r;               // normalized log odds ratio for relative
  float nor_c;               // complement of nor_r (for maxin learning)
  float act;                 // final activation (rebel prob.)
  float p_act;               // prior probability
  float da;                  // change in activity from one cycle
  float act_m;               // minus-phase activity value
  float act_p;               // plus-phase activity value

  Layer*     my_lay;         // layer this unit is in
  Connection bias;           // bias weight values
  Connection cons[];         // an array of receiving connections

  // functions
  Compute_Net_Input() {      // sender based is used in pdp++
    net = bias.wt;           // but recv based is shown here
    for Connection* c over cons {
      if(GAUSSIG) {
        net += (c->su->act * c->wt) *
                (1 - c->k_gaus * (c->su->act - c->wt)^2);
      }
      else
```

```
        net += c->su->act * c->wt;
    }
    // time-average net input
    net = prv_net + net_dt * (net - prv_net);
    prv_net = net;
  };
};

// defines the variables in a connection
class Connection {
  // parameters
  float k_gaus = 2.0;        // GAUSSIG weighting parameter
  float lrate = .01;         // learning rate
  float k_err = 1.0;         // strength of error-driven
  float k_assoc = .1;        // strength of associative
  float k_scl = .25;         // strength of soft-cmp-lrn asc

  // variables
  Unit* su;                  // pointer to the sending unit
  float wt;                  // weight value
  float dwt_err;             // change from error (chl)
  float dwt_asc;             // change from assoc (maxin)
  float dwt;                 // overall weight change

  // functions
  Compute_dWt(Unit* ru) {    // ru is receiving unit
    // first compute CHL error-driven component
    if(GAUSSIG) {
      float ru_act_dif = ru->act_p - ru->act_m;
      float su_act_dif = su->act_p - su->act_m;
      dwt_err = (ru->act_p * su->act_p) - (ru->act_m * su->act_m) +
       k_gaus * cn->wt *     // this extra term is for gaussig
        (ru_act_dif * (su->act_p * (su->act_p - wt) +
                       su->act_m * (su->act_m - wt)) +
         su_act_dif * (ru->act_p * (ru->act_p - wt) +
                       ru->act_m * (ru->act_m - wt)));
    }
    else
      dwt_err = (ru->act_p * su->act_p) - (ru->act_m * su->act_m);

    dwt = dwt_err * k_err;         // weight by err strength

    // now compute MaxIn associative component
    dwt_asc =                   //
     (ru->act_p * ru->nor_c * (su->act_p - su->my_lay->avg_act)) +
      (k_scl * ru->act_p * (su->act_p - wt));

    // only add assoc if err is positive or assoc is negative
    if((dwt >= 0) || (dwt_asc < 0))
      dwt += k_assoc * dwt_asc;
  };

  Update_Weights() {
    // first perform soft-bounding on the weights
```

```
    if(dwt > 0) dwt *= 1 - wt;
    else        dwt *= wt;

    wt += lrate * dwt;
  };
};
```

Derivation of GeneRec for the GausSig Function

As described in Chapter 4, the GeneRec learning rule needs to incorporate the derivative of the individual probability function with respect to the net input term to work with the GausSig function. While the standard GeneRec learning rule (e.g., for the CHL case) as derived in Chapter 2 does work, the correct learning rule for GausSig as derived here typically works better. The learning rule ends up being essentially CHL with an additional weight-based cost function, the biological plausibility of which is not clear, but it is locally computable and might correspond to some influence of the current weight value on the magnitude and direction of synaptic modification in cortical neurons.

There are two ways of approaching the use of GausSig in learning. One is to consider the actual derivative of the GausSig function with the activation and weight modulated distance term (4.29), which is rather complicated due to the multiple dependencies on the weight which is the variable of differentiation, and results in a complicated expression. The other is to consider the simpler GausSig function without the activation and weight multiplying the distance penalty, which results in a considerably simpler and cleaner learning rule. This latter approach is taken here, since it is the two magnitude and distance terms in GausSig which should be optimized by learning, not the activation and weight multiplying term.

The derivative of the GausSig function (4.29) with respect to the weight, and treating the activation and weight multiplier on the distance term as a constant, is (replacing k_{gauss} with k for simplicity):

$$\frac{\partial \eta}{\partial w_{ij}} = x_i + 2k x_i w_{ij}(x_i - w_{ij}) \tag{9.10}$$

which can be compared with the much simpler $\frac{\partial \eta}{\partial w_{ij}} = x_i$ for the standard dot product net input. Thus, the derivation of a symmetric, approximate midpoint version of the GeneRec learning rule (analogous to the CHL learning rule for dot-product based net inputs) is as follows:

$$\begin{aligned} \frac{1}{\epsilon}\Delta w_{ij} &= \frac{1}{2}\Big[(y_j^+ - y_j^-)[x_i^+ + 2k x_i^+ w_{ij}(x_i^+ - w_{ij}) + x_i^- + 2k x_i^- w_{ij}(x_i^- - w_{ij})] + \\ &\quad (x_i^+ - x_i^-)[y_j^+ + 2k y_j^+ w_{ij}(y_j^+ - w_{ij}) + y_j^- + 2k y_j^- w_{ij}(y_j^- - w_{ij})]\Big] \end{aligned} \tag{9.11}$$

which can be simplified as:

$$\frac{1}{\epsilon}\Delta w_{ij} = (x_i^+ y_j^+ - x_i^- y_j^-) + k w_{ij}\Big[(y_j^+ - y_j^-)[x_i^+(x_i^+ - w_{ij}) + x_i^-(x_i^- - w_{ij})] +$$

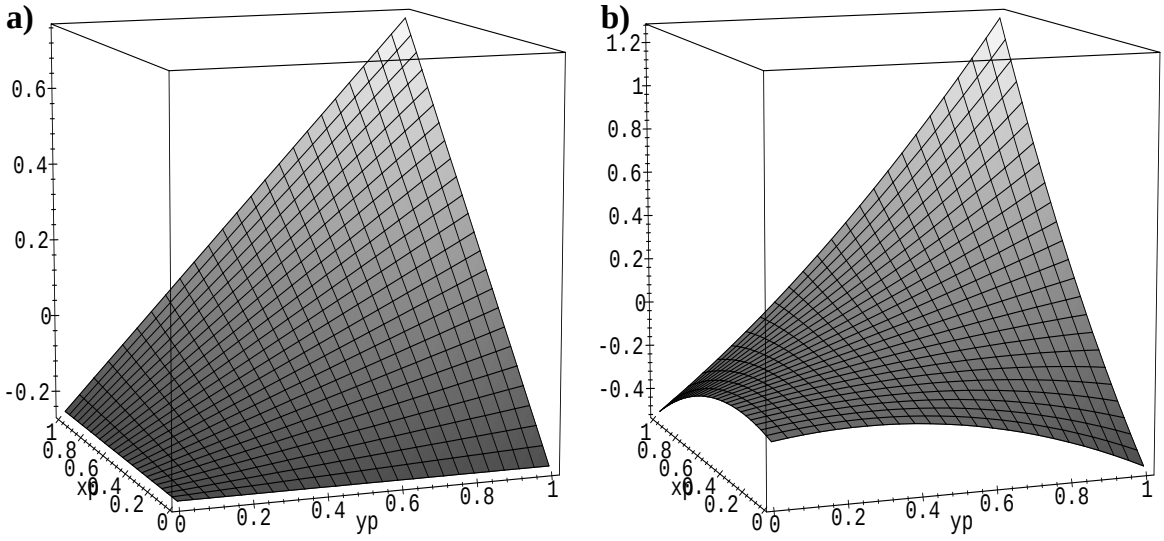

Figure 9.2: Differences between GausSig and standard sigmoidal version of the CHL learning rule. The computed weight change is shown as a function of the sending (xp) and receiving (yp) activations in the plus phase (for fixed minus phase values of .5 for both xm and ym), with a weight value of .5 and a k_gauss parameter of 2. The differences between the shape of these learning rules is not substantial, but does make some difference in learning rate, probably due to the larger value of the weight changes for GausSig.

$$(x_i^+ - x_i^-)[y_j^+(y_j^+ - w_{ij}) + y_j^-(y_j^- - w_{ij})]\Big] \tag{9.12}$$

The differences between this GausSig version of the CHL learning rule and the standard sigmoidal version can be seen in Figure 9.2, which shows how the computed weight change varies as a function of the sending and receiving activations in the plus phase. The differences are not substantial, but the GausSig version tends to learn more rapidly, which is probably due to its larger overall magnitude.

Guidelines for Setting Parameters in LEABRA

This section contains a set of guidelines distilled from my experience in getting LEABRA simulations to work well. The default parameters as described above work in a wide range of cases, but things change a bit when using very large networks, and networks with very different levels of activity in different layers.

Large Networks: For large networks, it is often necessary to sharpen the competition between hidden units, given the law-of-large-numbers effect which will make the variance around the mean for the net inputs to the units lower as the number of inputs is increased. This can happen in a network with more than around 100 or 200 hidden units. To compensate for the reduced variance, use a gain parameter γ for the normalization of the odds ratio in ReBel (which defines how "sharp" the competition is between units) of 2.25 instead of the usual 2. In addition, the level of activity in large hidden layers should typically be around 15%, as opposed to the 25% level that works best in smaller

networks. Finally, the k_{gauss} parameter should be decreased in large networks (from 2 to 1, for example), since the distances will be larger overall.

The balance between the two MaxIn learning components should also be altered for larger networks, which need to emphasize entropy reduction over information preservation since they have more degrees of freedom. Thus, the weighting of the SCL (soft-competitive learning) component, which performs information preservation, should be reduced from the standard value of .25 to something like .1. This should also be reduced in other cases where information preservation should be de-emphasized, such as with the random distributed input representations in the family trees task, as discussed in Chapter 7. It may also be important to reduce the overall magnitude of the associative learning component (k_{assoc}) in large networks.

Activity Levels: The weight gain parameters that control the overall gain of a set of inputs coming from a given layer must be set to compensate for the case when different layers have very different activity levels. This can be tricky, because an input contributes to which units are active in terms of the amount of *variance* it imparts across the different units in another layer, not in the raw net input level, which is normalized away in the soft kWTA ReBel function anyway (c.f., O'Reilly & McClelland, 1994). The actual level of variance depends on the number of active input units as well as the weight values from those units. As a general rule, if the numbers of active units are relatively close (e.g., 5 in one layer and 7 in another), then the weight gain can be left at one. However, if, for example, there are 12 active units in one layer and only 1 in another, then the weight gain from the layer with only one active input should be greater than 1 but less than 12. If you consider the variance to increase as the square root of the number of active units, then a value of roughly 3.5 should be used as a weight gain from the layer with one active unit. However, in practice it seems that $2\sqrt{N}$ gives better results for cases with only on active input. Thus, a value of roughly 7 should be used in this example.

The value of k_{gauss} in (4.29) is typically 2, except when the input is more sparse (or the input layer is large), in which case it is 1. If there is only one input present at any given time, then a value of 0 is appropriate, since the advantages of the GausSig function will not apply in this case.

For a layer with a very sparse activity level (below 10%), it can be important to lower the learning threshold for the bias and other weights in that layer from the standard of .1. For example, in the family trees network with orthogonal localist inputs, which have around 4% activity, a threshold of .02 was used. This enables the smaller overall activation differences in such layers to register weight changes more often.

Layers with higher levels of activity (above 30%) appear to impair learning and generalization performance in LEABRA. One reason for this is that the associative learning will tend to saturate with higher activity levels, since the baseline level of association will be quite high. Thus, such high activity levels should be avoided.

References

Abu-Mostafa, Y. S. (1989). The Vapnik-Chervonenkis dimension: Information versus complexity in learning. *Neural Computation, 1*, 312–317.

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science, 9*, 147–169.

Almeida, L. B. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In M. Caudil, & C. Butler (Eds.), *Proceedings of the IEEE First International Conference on Neural Networks San Diego, CA* (pp. 609–618).

An, G. (1996). The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation, 8*, 643–674.

Anderson, J. R. (1990). *The adaptive character of thought*. Hillsdale, NJ: Lawrence Earlbaum Associates, Inc.

Artola, A., Brocher, S., & Singer, W. (1990). Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex. *Nature, 347*, 69–72.

Artola, A., & Singer, W. (1993). Long-term depression of excitatory synaptic transmission and its relationship to long-term potentiation. *Trends In Neurosciences, 16*, 480.

Barlow, H. B. (1989). Unsupervised learning. *Neural Computation, 1*, 295–311.

Bashir, Z., Bortolotto, Z. A., & Davies, C. H. (1993). Induction of LTP in the hippocampus needs synaptic activation of glutamate metabotropic receptors. *Nature, 363*, 347–350.

Battiti, T. (1992). First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation, 4*(2), 141–166.

Bear, M. F., & Malenka, R. C. (1994). Synaptic plasticity: LTP and LTD. *Current Opinion in Neurobiology, 4*, 389–399.

Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *The Journal of Neuroscience, 2*(2), 32–48.

Brocher, S., Artola, A., & Singer, W. (1992). Intracellular injection of Ca2+ chelators blocks induction of long-term depression in rat visual cortex. *Proceedings of the National Academy of Sciences, 89*, 123.

Brousse, O. (1993). *Generativity and systematicity in neural network combinatorial learning* (Technical Report CU-CS-676-93). University of Colorado at Boulder, Department of Computer Science.

Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.

Cleeremans, A., Servan-Schreiber, D., & McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation, 1*, 372–381.

Cohen, J. D., Dunbar, K., & McClelland, J. L. (1990). On the control of automatic processes: A parallel distributed processing model of the stroop effect. *Psychological Review, 97*(3), 332–361.

Cohen, J. D., & O'Reilly, R. C. (1996). A preliminary theory of the interactions between prefrontal cortex and hippocampus that contribute to planning and prospective memory. In M. Brandimonte, G. O. Einstein, & M. A. McDaniel (Eds.), *Prospective memory: Theory and applications*. Hove, England: Lawrence Earlbaum Associates.

Cohen, J. D., & Servan-Schreiber, D. (1992). Context, cortex, and dopamine: A connectionist approach to behavior and biology in schizophrenia. *Psychological Review, 99*, 45–77.

Collingridge, G. L., & Bliss, T. V. P. (1987). NMDA receptors - their role in long-term potentiation. *Trends In Neurosciences, 10*, 288–293.

Creutzfeldt, O. D. (1977). Generality of the functional structure of the neocortex. *Naturwissenshaften, 64*, 507–517.

Crick, F. H. C. (1989). The recent excitement about neural networks. *Nature, 337*, 129–132.

Crick, F. H. C., & Mitchison, G. (1983). The function of dream sleep. *Nature, 304*, 111–114.

Daugherty, K. (1993). *A connectionist approach to the past tense of English*. PhD thesis, Department of Computer Science, University of Southern California, Los Angeles, CA.

Dayan, P., Hinton, G. E., Neal, R. N., & Zemel, R. S. (1995). The helmholtz machine. *Neural Computation, 7*, 889–904.

Dayan, P., & Zemel, R. S. (1995). Competition and multiple cause models. *Neural Computation, 7*, 565–579.

Desimone, R., & Ungerleider, L. G. (1989). Neural mechanisms of visual processing in monkeys. In F. Boller, & J. Grafman (Eds.), *Handbook of neurophysiology, vol. 2* (Chap. 14, pp. 267–299). Amsterdam: Elsevier.

Douglas, R. J., Koch, C., Mahowald, M., Martin, K. A. C., & Suarez, H. H. (1995). Recurrent excitation in neocortical circuits. *Science, 269*, 981.

Douglas, R. J., & Martin, K. A. C. (1990). Neocortex. In G. M. Shepherd (Ed.), *The synaptic organization of the brain* (Chap. 12, pp. 389–438). Oxford: Oxford University Press.

Douglas, R. J., Martin, K. A. C., & Whitteridge, D. (1989). A cannonical microcircuit for neocortex. *Neural Computation, 1*, 480–488.

Duhamel, J.-R., Colby, C. L., & Goldberg, M. E. (1992). The updating of the representation of visual space in parietal cortex by intended eye movements. *Science, 255*, 90–92.

Felleman, D. J., & Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex, 1*, 1–47.

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition, 28*, 3–71.

Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics, 64*(2), 165–170.

Földiák, P. (1991). Learning invariance from transformation sequences. *Neural Computation, 3*(2), 194–200.

Galland, C. C. (1993). The limitations of deterministic Boltzmann machine learning. *Network, 4*, 355–379.

Galland, C. C., & Hinton, G. E. (1990). Discovering high order features with mean field modules. In (Touretzky, 1990).

Galland, C. C., & Hinton, G. E. (1991). Deterministic Boltzmann learning in networks with asymmetric connectivity. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, & G. E. Hinton (Eds.), *Connectionist Models: Proceedings of the 1990 Summer School* (pp. 3–9). San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Geman, S., Bienenstock, E. L., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1–58.

Georgopoulos, A. P. (1990). Neurophysiology and reaching. In M. Jeannerod (Ed.), *Attention and performance, vol. 13* (pp. 227–263). Hillsdale, N.J.: Erlbaum.

Gershberg, F. B., & Shimamura, A. P. (1995). Impaired use of organizational strategies in free recall following frontal lobe damage. *Neuropsychologia, 33*, 1305.

Gibson, W. G., Robinson, J., & Bennett, M. R. (1991). Probabilistic secretion of quanta in the central nervous system: Granule cell synaptic control of pattern separation and activity regulation. *Philosophical Transactions of the Royal Society (Lond.) B, 332*, 199–220.

Goodhill, G. J., & Barrow, H. G. (1994). The role of weight normalization in competitive learning. *Neural Computation, 6*, 255–269.

Hancock, P. J. B., Smith, L. S., & Phillips, W. A. (1991). A biologically supported error-correcting learning rule. *Neural Computation, 3*, 201–212.

Hillyard, S. A., & Picton, T. W. (1987). Electrophysiology of cognition. In F. Plum (Ed.), *Handbook of physiology, section I: Neurophysiology volume V: Higher functions of the brain* (pp. 519–584). American Physiological Society.

Hinton, G. E. (1986). Learning distributed representations of concepts. *Proceedings of the 8 th Conference of the Cognitive Science Society* (pp. 1–12). Hillsdale, NJ: Lawrence Earlbaum Associates, Inc.

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

Hinton, G. E. (1989b). Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation, 1*, 143–150.

Hinton, G. E., Dayan, P., Frey, B. J., & Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science, 268*, 1158.

Hinton, G. E., & McClelland, J. L. (1988). Learning representations by recirculation. In D. Z. Anderson (Ed.), *Neural Information Processing Systems, 1987* (pp. 358–366). New York: American Institute of Physics.

Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). Distributed representations. In (Rumelhart, McClelland, & PDP Research Group, 1986b), Chap. 3, pp. 77–109.

Hinton, G. E., & Sejnowski, T. J. (1983, June). Optimal perceptual inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Washington, DC.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In (Rumelhart et al., 1986b), Chap. 7, pp. 282–317.

Hirsh, J. C., & Crepel, F. (1992). Postsynaptic Ca^{2+} is necessary for the induction of LTP and LTD of monosynaptic EPSPs in prefrontal neurons: An in vitro study in the rat. *Synapse, 10*, 173–175.

Hoeffner, J. H. (1996). *A single mechanism account of the acquisition and processing of regular and irregular inflectional morphology*. PhD thesis, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA, 79*, 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA, 81*, 3088–3092.

Hubel, D. H., & Wiesel, T. N. (1965). Binocular interactions in striate cortex of kittens reared with artifical squint. *Journal of Neurophysiology, 28*, 1041–1059.

Hubel, D. H., & Wiesel, T. N. (1970). The period of susceptibility to the physiological effects of unilateral eye closure in kittens. *Journal of Physiology, 206*, 419–436.

Hubel, D. H., Wiesel, T. N., & LeVay, S. (1977). Plasticity of ocular dominance columns in monkey striate cortex. *Philosophical Transactions of the Royal Society, London. B., 278*, 377–409.

Jacobs, R. A. (1987). *Increased rates of convergence through learning rate adaptation* (COINS 87-117). Dept of Computer & Information Science, University of Massachussetts. delta-bar-delta.

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

Jaffe, D. B., Johnston, D., Lasser-Ross, N., Lisman, J. E., Miyakawa, H., & Ross, W. N. (1992). The spread of Na^{+} spikes determines the pattern of dendritic Ca^{2+} entry into hippocampal neurons. *Nature, 357*, 244–246.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*(2), 181–214.

Kaas, J. H. (1991). Plasticity of sensory and motor maps in adult mammals. *Annual Review of Neuroscience*, *14*, 137–167.

Karmiloff-Smith, A. (1992). *Beyond modularity: A developmental perspective on cognitive science*. Cambridge, MA: MIT Press.

Kass, R. E., & Raftery, A. E. (1993). *Bayes factors and model uncertainty* (Technical Report 571). Pittsburgh, PA 15213: Carnegie Mellon University Dept of Statistics.

Kucera, H., & Francis, W. (1967). *Computational analysis of present-day American English*. Providence, RI: Brown University Press.

Kullmann, D. M., Perkel, D. J., Manabe, T., & Nicoll, R. A. (1992). Ca^{2+} entry via postsynaptic voltage-sensitive Ca^{2+} channels can transiently potentiate excitatory synaptic transmission in the hippocampus. *Neuron*, *9*, 1175–1183.

LeCun, Y., & Denker, J. S. (1991). A new learning rule for recurrent networks. *Proceedings of of the Conference on Neural Networks for Computing (Snowbird, UT)*.

Levitt, J. B., Lewis, D. A., Yoshioka, T., & Lund, J. S. (1993). Topography of pyramidal neuron intrinsic connections in macaque monkey prefrontal cortex (areas 9 & 46). *Journal of Comparative Neurology*, *338*, 360–376.

Linden, D. J. (1994). Long-term synaptic depression in the mammalian brain. *Neuron*, *12*, 457–472.

Linsker, R. (1988). Self-organization in a perceptual network. *Computer*, 105–117.

Linsker, R. (1992). Local synaptic learning rules suffice to maximize mutual information in a linear network. *Neural Computation*, *4*, 691–702.

Lisman, J. (1994). The CaM Kinase II hypothesis for the storage of synaptic memory. *Trends in neurosciences*, *17*, 406.

Lisman, J. E. (1989). A mechanism for the hebb and the anti-hebb processes underlying learning and memory. *Proc. Natl. Acad. Sci. USA*, *86*, 9574–9578.

Malenka, R. C., Lancaster, B., & Zucker, R. S. (1992). Temporal limits on the rise in postsynpatic calcium required for the induction of long-term potentiation. *Neuron*, *9*, 121–128.

Malenka, R. C., & Nicoll, R. A. (1993). NMDA receptor-dependent synaptic plasticity: Multiple forms and mechanisms. *Trends In Neurocience*, *16*, 521–527.

Marcus, G. F., Pinker, S., Ullman, M., Hollander, M., Rosen, J. T., & Xu, F. (1992). Overregularization in language acquisition. *Monographs of the Society for Research in Child Development*, *57*(4), 1–165.

Marr, D. (1982). *Vision*. New York: Freeman.

Maunsell, J. H. R., & Newsome, W. T. (1987). Visual processing in monkey extrastriate cortex. *Annual Review of Neuroscience, 10*, 363–401.

Mazzoni, P., Andersen, R. A., & Jordan, M. I. (1991). A more biologically plausible learning rule for neural networks. *Proc. Natl. Acad. Sci. USA, 88*, 4433–4437.

McClelland, J. L. (1993). The GRAIN model: A framework for modeling the dynamics of information processing. In D. E. Meyer, & S. Kornblum (Eds.), *Attention and performance XIV: Synergies in experimental psychology, artifical intelligence, and cognitive neuroscience* (pp. 655–688). Hillsdale, NJ: Lawrence Erlbaum Associates.

McClelland, J. L. (1994). The interaction of nature and nurture in development: A parallel distributed processing perspective. In P. Bertelson, P. Eelen, & G. D'Ydewalle (Eds.), *Current advances in psychological science: Ongoing research* (pp. 57–88). Hillsdale, NJ: Erlbaum.

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionst models of learning and memory. *Psychological Review, 102*, 419–457.

McClelland, J. L., & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part 1. an account of basic findings. *Psychological Review, 88*(5), 375–407.

McClelland, J. L., & Rumelhart, D. E. (Eds.). (1988). *Explorations in parallel distributed processing: a handbook of models, programs, and exercises*. Cambridge, MA: MIT Press.

McCloskey, M. (1991). Networks and theories: The place of connectionism in cognitive science. *Psychological Science, 2*, 387–395.

McNaughton, B. L., & Morris, R. G. M. (1987). Hippocampal synaptic enhancement and information storage within a distributed memory system. *Trends In Neurosciences, 10*(10), 408–415.

Merzenich, M. M., & Sameshima, K. (1993). Cortical plasticity and memory. *Current Opinion in Neurobiology, 3*, 187–196.

Miller, K. D., Keller, J. B., & Stryker, M. P. (1989). Ocular dominance column development: Analysis and simulation. *Science, 245*, 605–615.

Miller, K. D., & MacKay, D. J. C. (1994). The role of constraints in Hebbian learning. *Neural Computation, 6*, 100–126.

Montague, P. R., Dayan, P., & Sejnowski, T. J. (1996). a framework for mesencephalic dopamine systems based on predictive hebbian learning. *Journal of Neuroscience, 16*, 1936.

Movellan, J. R. (1990). Contrastive Hebbian learning in the continuous Hopfield model. In D. S. Touretzky, G. E. Hinton, & T. J. Sejnowski (Eds.), *Proceedings of the 1989 Connectionist Models Summer School* (pp. 10–17). San Mateo, CA: Morgan Kaufmann.

Movellan, J. R., & McClelland, J. L. (1993). Learning continuous probability distributions with symmetric diffusion networks. *Cognitive Science, 17*, 463–496.

Mulkey, R. M., & Malenka, R. C. (1992). Mechanisims underlying induction of homosynaptic long-term depression in area CA1 of the hippocampus. *Science, 9*, 967–975.

Nowlan, S. J. (1990). Maximum likelihood competitive learning. In (Touretzky, 1990).

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology, 15*, 267–273.

O'Reilly, R. C. (1994). Temporally local unsupervised learning: The MaxIn algorithm for maximizing input information. In M. C. Mozer, P. Smolensky, & A. S. Weigend (Eds.), *Proceedings of the 1993 Connectionist Models Summer School*. Hove, England: Lawrence Earlbaum Associates.

O'Reilly, R. C., & Johnson, M. H. (1994). Object recognition and sensitive periods: A computational analysis of visual imprinting. *Neural Computation*, *6*(3), 357–389.

O'Reilly, R. C., & McClelland, J. L. (1992). *The self-organization of spatially invariant representations* (Parallel Distributed Processing and Cognitive Neuroscience PDP. CNS. 92. 5). Carnegie Mellon University, Department of Psychology.

O'Reilly, R. C., & McClelland, J. L. (1994). Hippocampal conjunctive encoding, storage, and recall: Avoiding a tradeoff. *Hipocampus*, *4*(6), 661–682.

Pearl, J. (1988). *Probabalistic reasoning in intelligent systems*. San Mateo: Morgan Kaufman.

Pearlmutter, B. A., & Hinton, G. E. (1986). G-maximization: An unsupervised learning procedure for discovering regularities. In J. S. Denker (Ed.), *Neural networks for computing* (pp. 333–338). New York: American Institute of Physics.

Perkel, D. J., Petrozzino, J. J., Nicoll, R. A., & Connor, J. A. (1993). The role of Ca^{2+} entry via synaptically activated NMDA receptors in the induction of long-term potentiation. *Neuron*, *11*, 817–823.

Peterson, C. (1991). Mean field theory neural networks for feature recognition, content addressable memory, and optimization. *Connection Science, 3*, 3–33.

Peterson, C., & Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, *1*, 995–1019.

Peterson, C., & Hartman, E. (1989). Explorations of the mean field theory learning algorithm. *Neural Networks*, *2*, 475–494.

Pineda, F. J. (1987a). Generalization of backpropagation to recurrent and higher order neural networks. In D. Z. Anderson (Ed.), *Proceedings of IEEE Conference on Neural Information Processing Systems, Denver, CO* (pp. 602–611). New York: IEEE.

Pineda, F. J. (1987b). Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, *18*, 2229–2232.

Pineda, F. J. (1988). Dynamics and architecture for neural computation. *Journal of Complexity*, *4*, 216–245.

Pinker, S., & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition, 28*, 73–193.

Plaut, D. C., McClelland, J. L., Seidenberg, M. S., & Patterson, K. E. (1996). Understanding normal and impaired word reading: Computational principles in quasi-regular domains. *Psychological Review, 103*, 56.

Plaut, D. C., & Shallice, T. (1993). Deep dyslexia: A case study of connectionist neuropsychology. *Cognitive Neuropsychology*, *10*(5), 377–500.

Plunkett, K., & Marchman, V. A. (1991). U-shaped learning and frequency effects in a multi-layered perceptron: Implications for child language acquisition. *Cognition, 38*, 43–102.

Plunkett, K., & Marchman, V. A. (1993). From rote learning to system building: Acquiring verb morphology in children and connectionist nets. *Cognition, 48*(1), 21–69.

Poggio, T., Torre, V., & Koch, C. (1985). Computational vision and regularization theory. *Nature, 317*, 314–319.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1988). *Numerical recipies in C: The art of scientific computing*. Cambridge: Cambridge University Press.

Rissanen, J. (1986). Stochastic complexity. *Annals of Statistics*, *14*, 1080–1100.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986a). Learning internal representations by error propagation. In (Rumelhart et al., 1986b), Chap. 8, pp. 318–362.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In J. L. McClelland, D. E. Rumelhart, & PDP Research Group (Eds.), *Parallel distributed processing. volume 2: Psychological and biological models* (pp. 216–271). Cambridge, MA: MIT Press.

Rumelhart, D. E., McClelland, J. L., & PDP Research Group (Eds.). (1986b). *Parallel distributed processing. volume 1: Foundations*. Cambridge, MA: MIT Press.

Rumelhart, D. E., & Zipser, D. (1986). Feature discovery by competitive learning. In (Rumelhart et al., 1986b), Chap. 5, pp. 151–193.

Saund, E. (1994). Unsupervised learning of mixtures of multiple causes in binary data. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances In Neural Information Processing Systems, 6*. San Mateo, CA: Morgan Kaufman.

Saund, E. (1995). A multiple cause mixture model for unsupervised learning. *Neural Computation*.

Schraudolph, N. N., & Sejnowski, T. J. (1996). Tempering backpropagation networks: Not all weights are created equal. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances In Neural Information Processing Systems, 8*. Cambridge, MA: MIT Press.

Schultz, W., Apicella, P., & Ljungberg, T. (1993). Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *The Journal of Neuroscience, 13*, 900–913.

Seidenberg, M. S., & McClelland, J. L. (1989). A distributed, developmental model of word recognition and naming. *Psychological Review, 96*, 523–568.

Shepherd, G. M. (1988). A basic circuit for cortical organization. In M. C. Gazzaniga (Ed.), *Perspectives on memory research* (pp. 93–134). Cambridge, MA: MIT Press.

Spelke, E., Breinlinger, K., Macomber, J., & Jacobson, K. (1992). Origins of knowledge. *Psychological Review, 99*, 605–632.

Stark, C. (1993). *Priming in a connectionist model of amnesia*.

Stryker, M. P., & Harris, W. A. (1986). Binocular impulse blockade prevents the formation of ocular dominance columns in cat visual cortex. *Journal of Neuroscience, 6*, 2117–2133.

Sur, M., Garraghty, P., & Roe, A. W. (1988). Experimentally induced visual projections into auditory thalamus and cortex. *Science, 242*, 1437–1441.

Sutton, S., Braren, M., Zubin, J., & John, E. R. (1965). Evoked-potential correlates of stimulus uncertainty. *Science, 150*, 1187–1188.

Szentágothai, J. (1978). The neuron network of the cerebral cortex: A functional interpretation. *Proceedings of the Royal Society (London) B, 201*, 219–248.

Tesauro, G. (1990). Neural models of classical conditioning: A theoretical viewpoint. In S. J. Hanson, & C. R. Olson (Eds.), *Connectionist modelling and brain function*. Cambridge, MA: MIT Press.

Torioka, T. (1979). Pattern separability in a random neural net with inhibitory connections. *Biological Cybernetics, 34*, 53–62.

Touretzky, D. S. (Ed.). (1990). *Advances in neural information processing systems, 2*. San Mateo, CA: Morgan Kaufmann.

Van Essen, D. C., & Maunsell, J. H. R. (1983). Hierarchical organization and functional streams in the visual cortex. *Trends In Neurosciences, 6*, 370–375.

Vecera, S. P., & O'Reilly, R. C. (submitted). Figure-ground organization and object recognition processes: An interactive account. *Journal of Experimental Psychology: Human Perception and Performance*.

von der Malsburg, C. (1973). Self-organization of orientation-sensitive columns in the striate cortex. *Kybernetik, 14*, 85–100.

Weigand, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forcasting. In R. P. Lippmann, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in neural information processing systems, 3*. San Mateo, CA: Morgan Kaufman.

Wolpert, D. H. (1992). On the connection between in-sample testing and generalization error. *Complex Systems, 6*, 47–94.

Zemel, R. S. (1993). *A minimum description length framework for unsupervised learning*. PhD thesis, University of Toronto, Canada.

Zipser, D., & Andersen, R. A. (1988). A back propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, *331*, 679–684.

Zipser, D., & Rumelhart, D. E. (1990). Neurobiologcial significance of new learning models. In E. Schwartz (Ed.), *Computational neuroscience* (pp. 192–200). Cambridge, MA: MIT Press.