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-

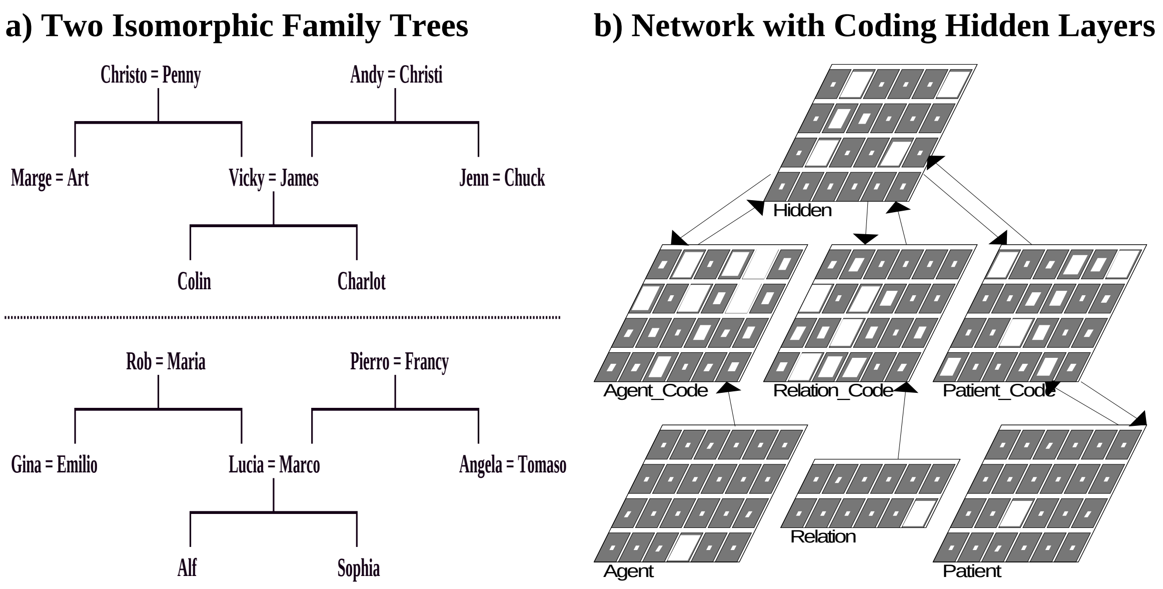


Figure 7.1: The family trees problem and network. In this network, the individual family members are represented both in the Agent and Patient layers as individual units. A version with distributed representations was also used. The code layers are supposed to develop activity patterns which capture 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

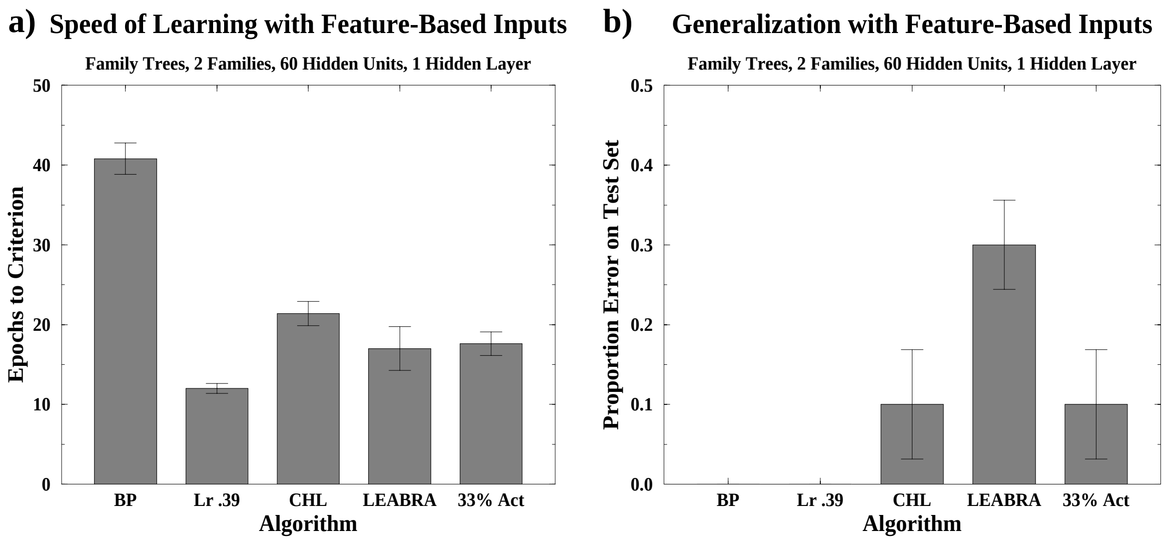


Figure 7.2: **a)** Learning speed and **b)** Generalization using feature-based inputs and a single hidden layer. **Lr .39** is BP with the "optimal" learning rate of .39. **33%** is LEABRA with 33% hidden layer activity level instead of the standard 25%, which improves generalization significantly.

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

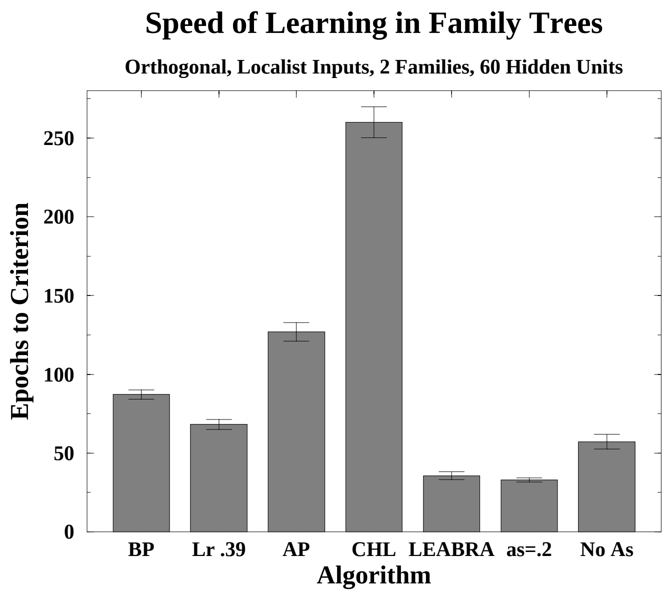


Figure 7.3: Learning speed for family trees task with orthogonal localist inputs. For BP, AP, and CHL, the fast learning rate of .1 was used, while LEABRA used the standard .01 learning rate. The **BP Lr .39** case was BP with 60 hidden units at the optimal learning rate (.39) for this task as found 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.

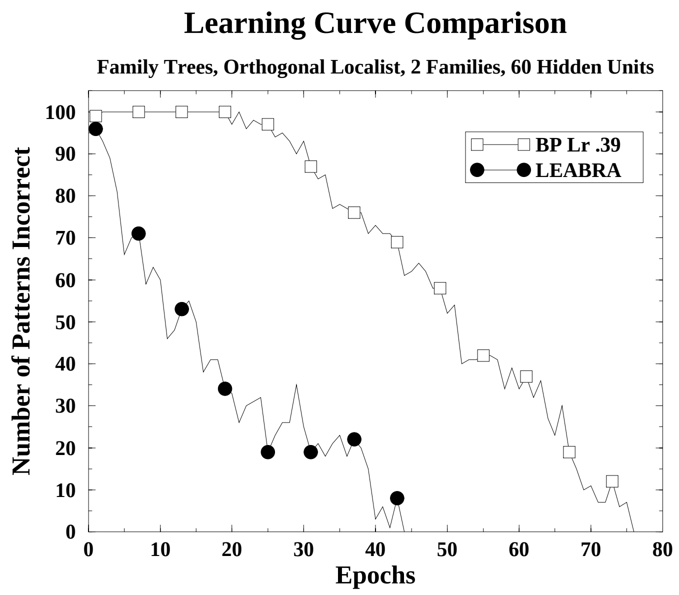


Figure 7.4: Learning curve from a single training run of BP with learning rate of .39, and standard LEABRA, using orthogonal localist inputs and 60 hidden units. BP has an initial plateau due to symmetry of error signals, while LEABRA learns rapidly from the start.

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

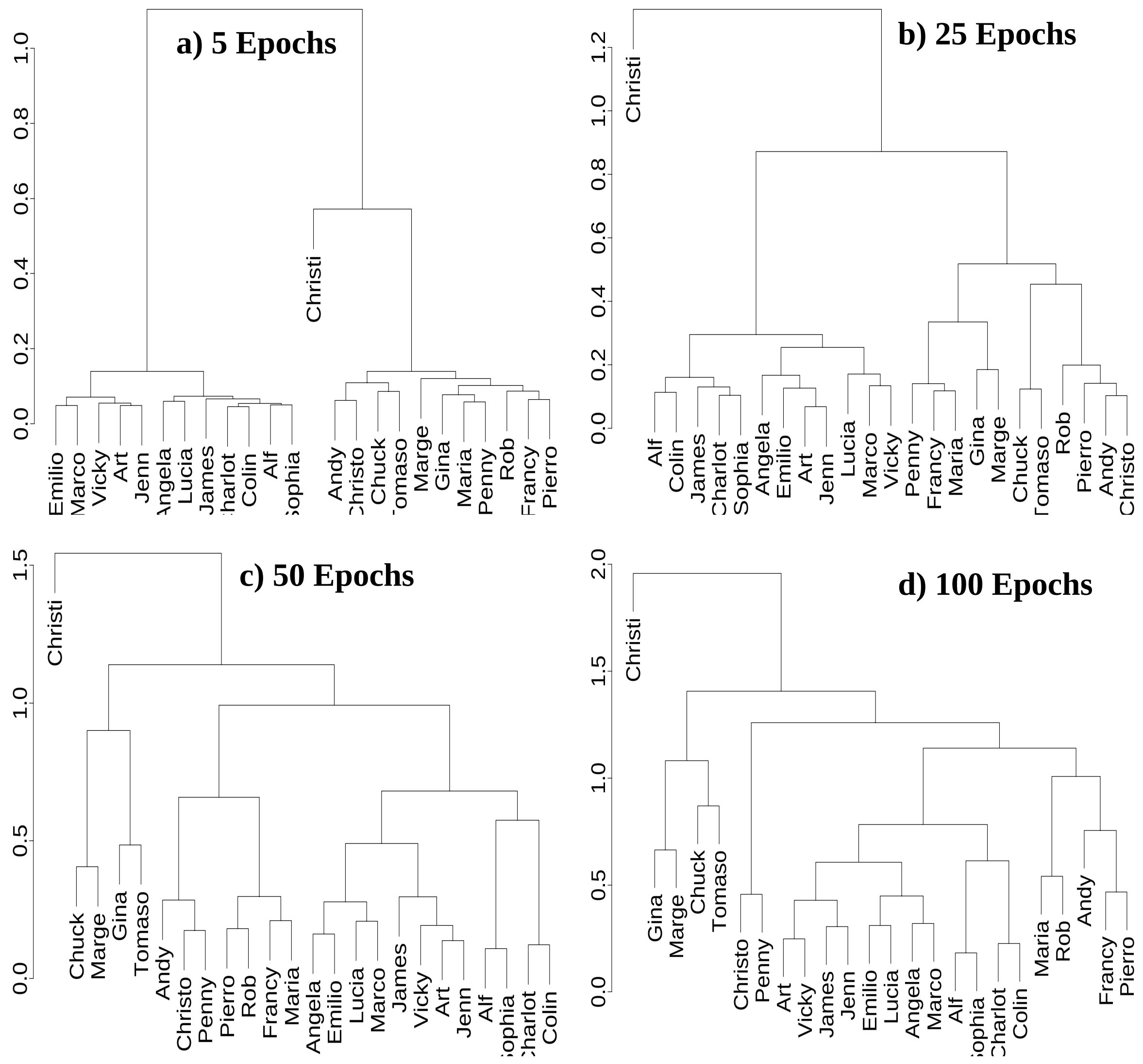


Figure 7.5: Development of representations in BP over time in terms of a cluster plot of the similarity structure of hidden unit representations (central hidden layer) for the Agents shown. Clustering was computed on the OR (max) of the activity values for training patterns in which the given agent appeared. Early in training (5 and 25 epochs), there is relatively little differentiation.

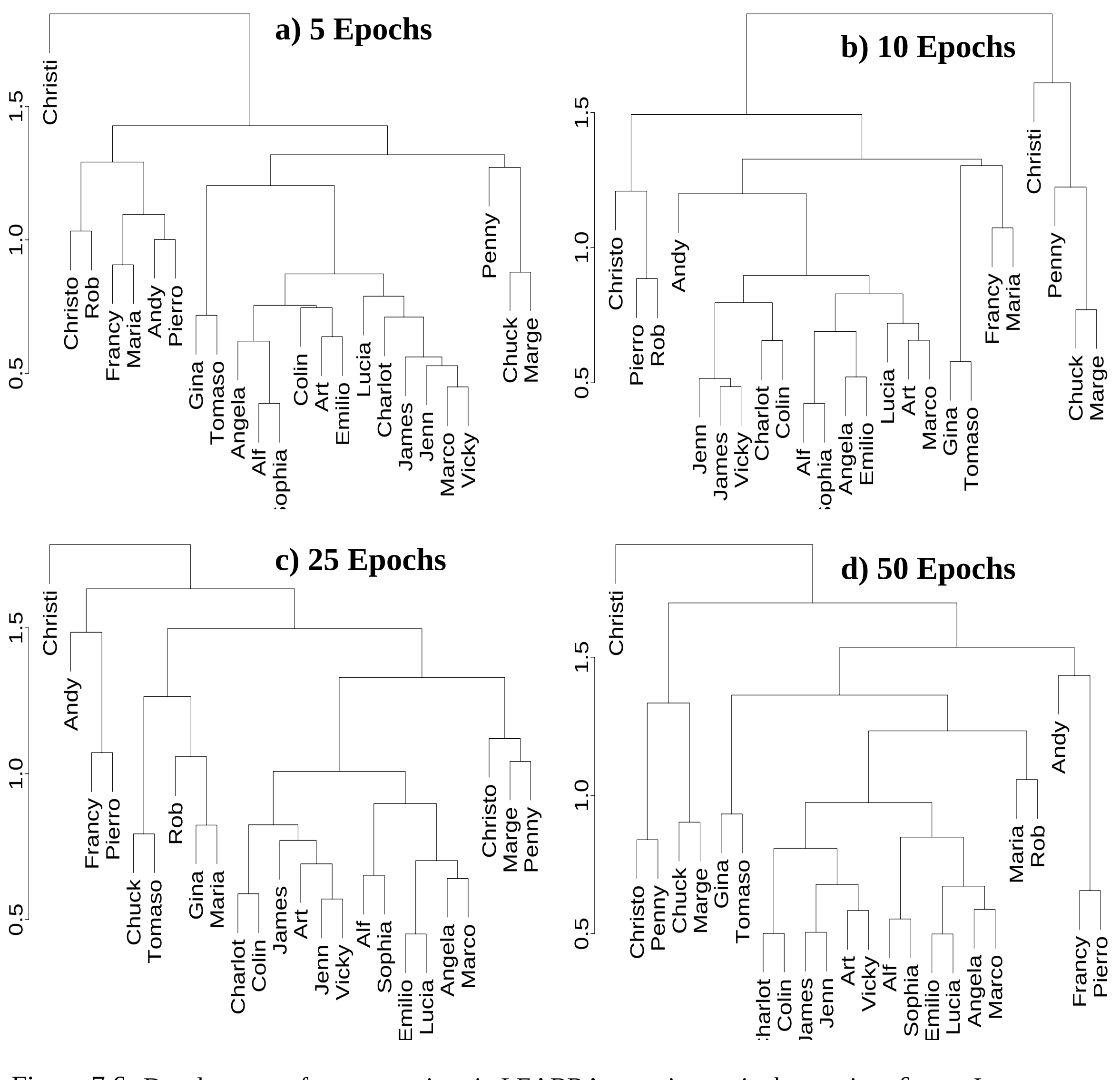


Figure 7.6: Development of representations in LEABRA over time as in the previous figure. In contrast to BP, considerable structure is evident after just 5 epochs of training.

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-

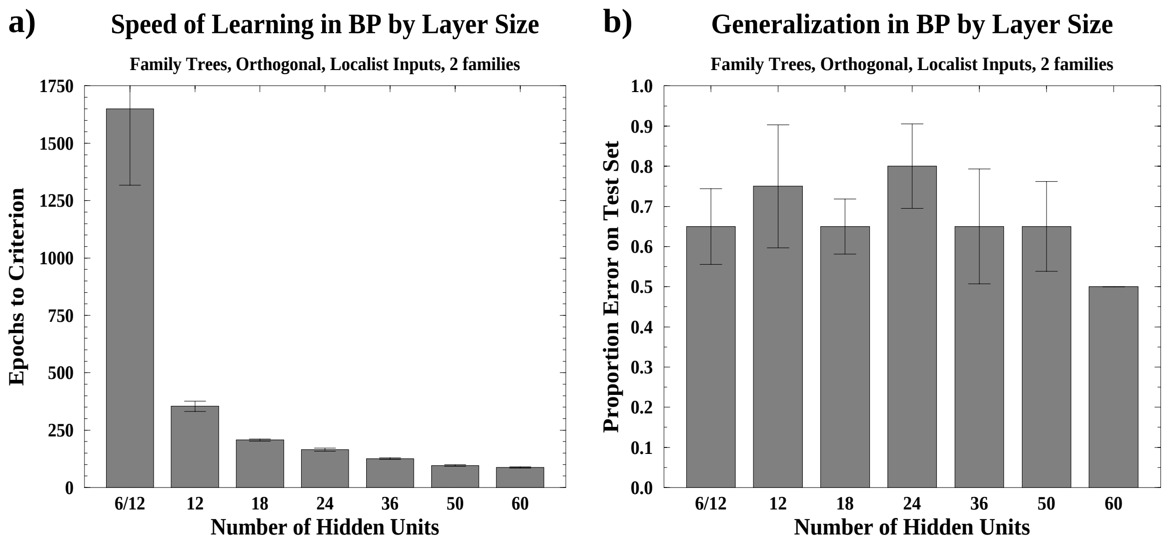


Figure 7.7: Effect of number of hidden units on **a)** Learning speed and **b)** Generalization performance in BP. The **6/12** case is 6 coding and 12 association hidden units, as used in the original Hinton, 1986 model. The remainder are the numbers of both coding and association hidden units. Learning 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-

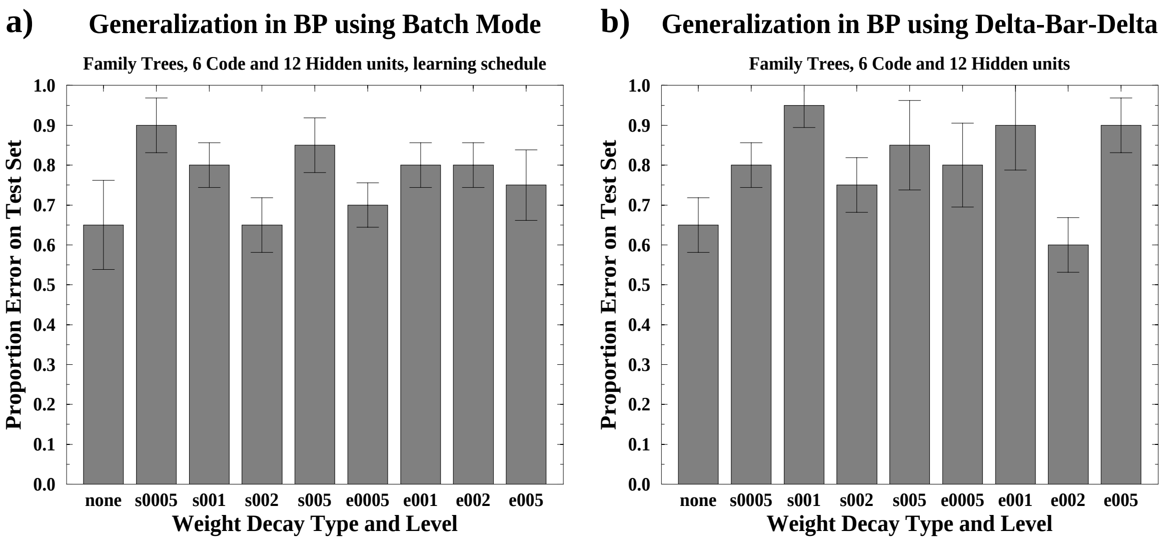


Figure 7.8: Generalization performance for batch mode learning, using **a)** a learning rate schedule and **b)** delta-bar-delta learning rate adaptation. **s** weight decay is simple weight decay (SWD), and **e** weight decay is weight elimination (WED). Results do not represent an improvement over 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.

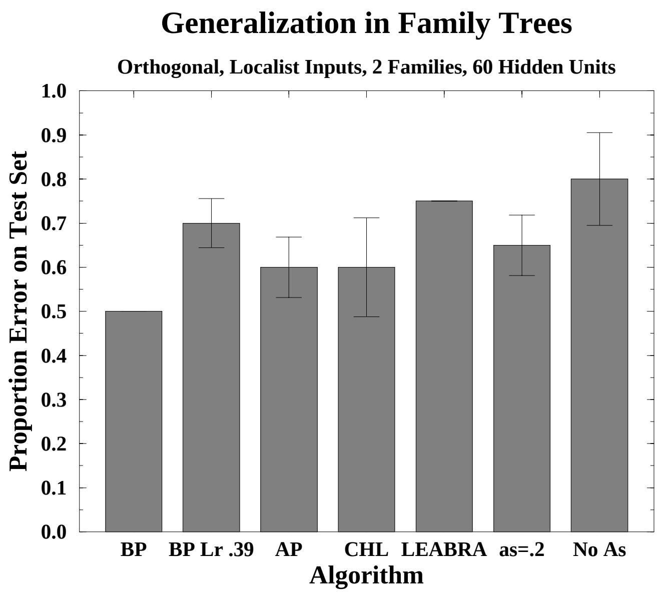


Figure 7.9: Generalization performance for various algorithms tested. The algorithms are as described in Figure 7.3.

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

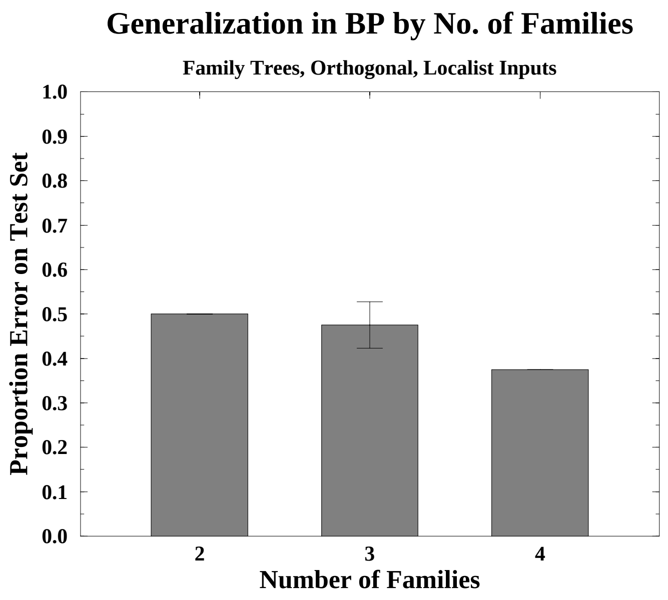


Figure 7.10: Effects of the number of families (training set size) on generalization performance in BP. While there appears to be some improvement in generalization for the 3 and 4 family sets, it is not clearly monotonically increasing with the number of families, and does not represent a substantial absolute improvement.

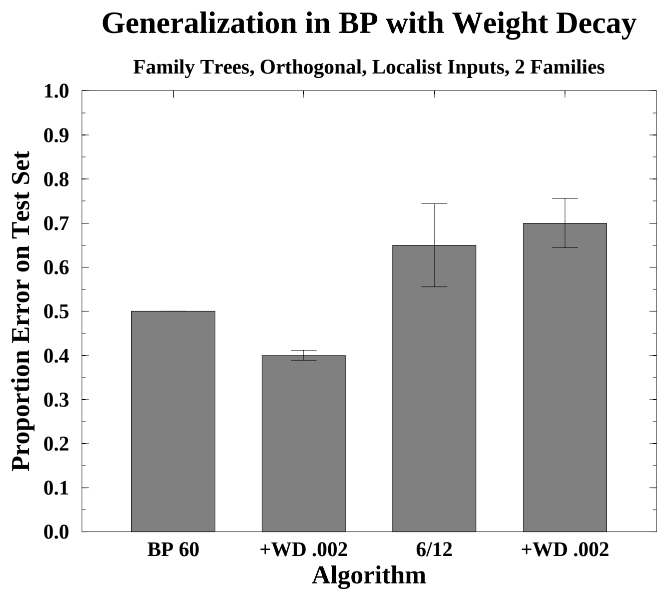


Figure 7.11: The effects of weight decay on generalization performance in BP. **+WD .002** is the previous condition plus .002 weight elimination. Note that .005 weight elimination prevented the networks from reaching criterion on training in all cases. **BP 60** is the standard network with 60 hidden 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,

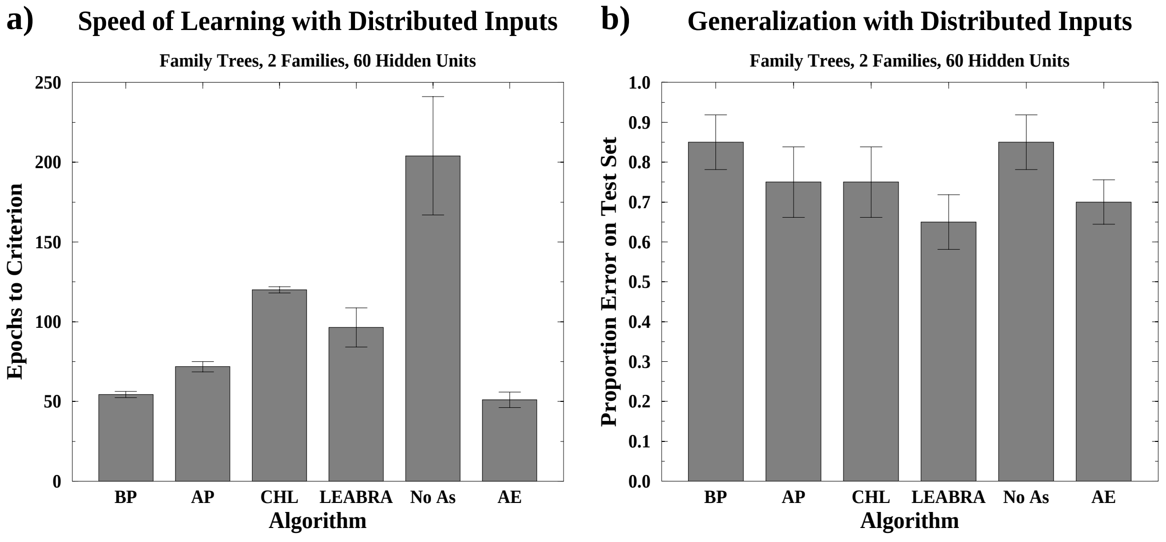


Figure 7.12: **a)** Learning speed and **b)** Generalization performance using random, distributed input patterns instead of the local, orthogonal ones. Algorithms are as in previous figures, with the addition of **AE** which is the LEABRA auto-associator model.

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

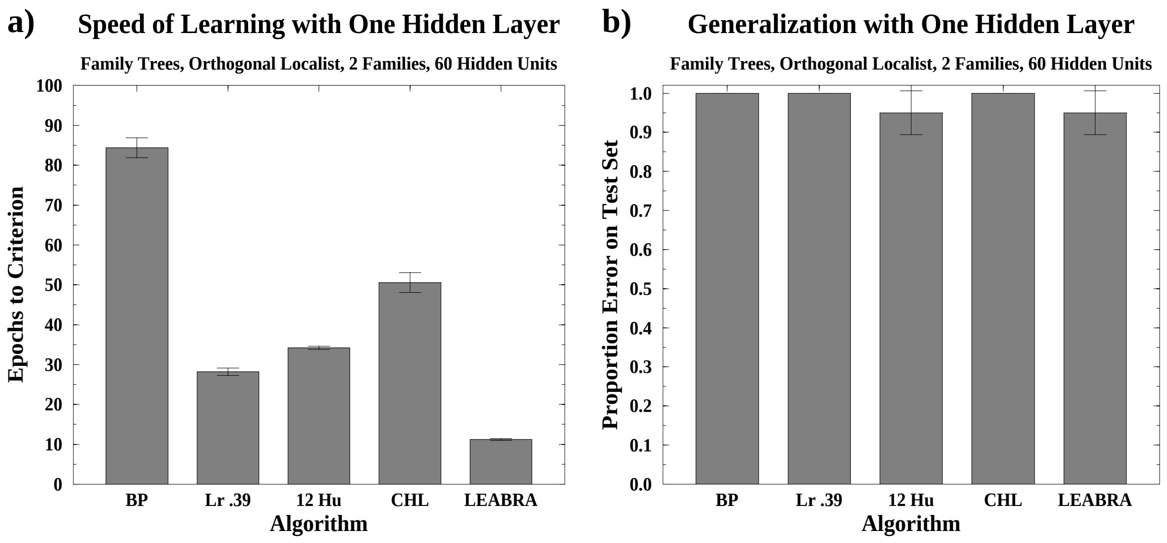


Figure 7.13: **a)** Learning speed and **b)** Generalization using orthogonal, localist inputs and a single hidden layer with 60 units (unless otherwise noted). **Lr .39** is BP with the optimal .39 learning rate. **12 Hu** is BP with 12 hidden units and the .39 learning rate.

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-

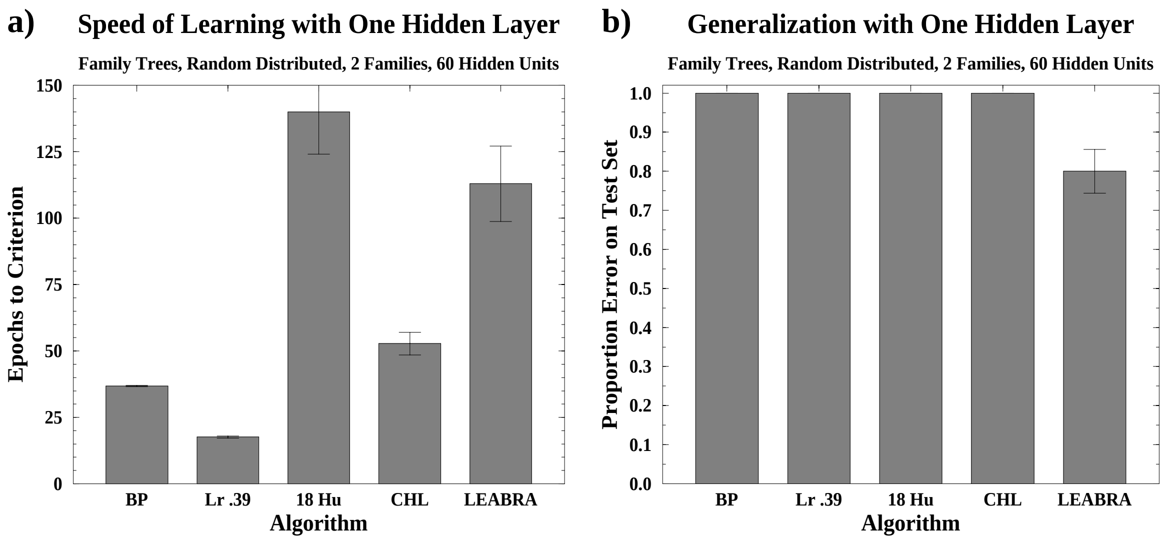


Figure 7.14: **a)** Learning speed and **b)** Generalization using random, distributed inputs and a single hidden layer. **Lr .39** is BP with the optimal .39 learning rate. **18 Hu** is BP with 18 hidden units and the .1 learning rate (could not learn with .39 learning rate).

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.

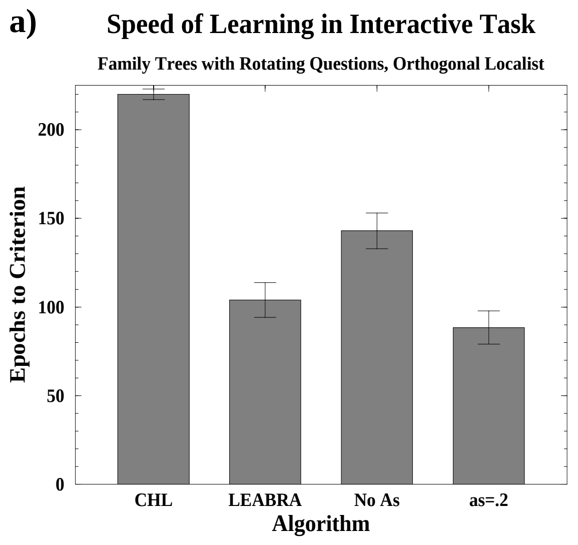
The generalization picture, also shown in Figure 7.14, is pretty much as expected, except for the remarkable results for LEABRA. Thus, both BP and CHL do not generalize at all, even with the bottleneck in BP. LEABRA, however, reliably generalized to one out of the 4 test cases in 4 out of the 5 networks 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



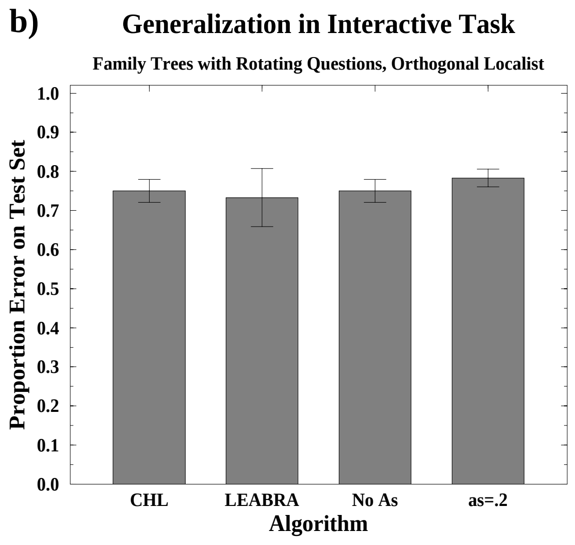


Figure 7.15: **a)** Learning speed and **b)** Generalization in an interactive version of the family trees task where the inputs and questions are rotated on each trial. **No As** is LEABRA without associative learning, and **as=.2** is with .2 associative learning.

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

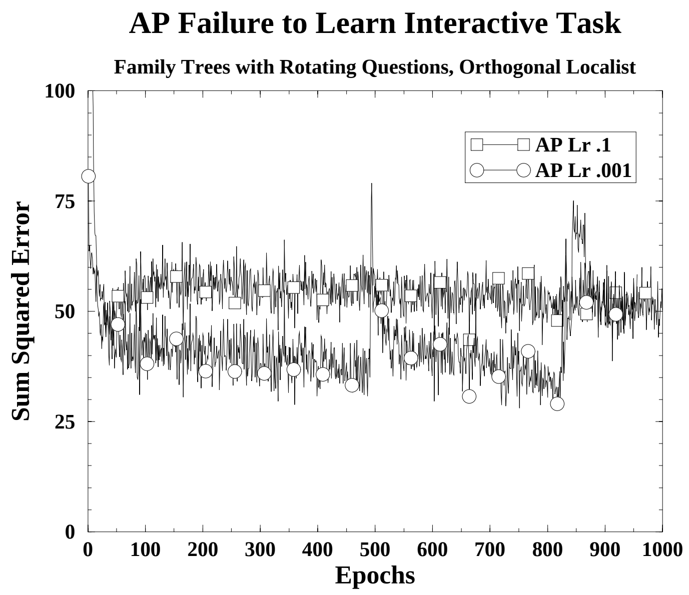


Figure 7.16: Learning curves for AP algorithm in an interactive version of the family trees task where the inputs and questions are rotated on each trial. Networks were unable to learn the task using a wide range of learning rates (.1, .001 shown, .02 also tried with similar results).

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.