Conjectures on Representations in Backpropagation Networks

Paul W. Munro[1]

TR-89-035

May, 1989

Abstract

The pros and cons of the backpropagation learning procedure have been the subject of numerous debates recently. Some point out its promise as a powerful instrument for finding the weights in a connectionist network appropriate to a given problem, and the generalizability of the solution to novel patterns. Others claim that it is an algorithm for fitting data to a function by error correction through gradient descent. The arguments in this paper focus on the latter (curve-fitting) point of view, but take the point of view that the power of back propagation comes from carefully choosing the form of the function to be fit. This amounts to choosing the architecture and the activation functions of the units (nodes) in the net. A discussion of the role of these two network features motivates two conjectures identifying the form of the squashing function as an important factor in the process. Some preliminary simulations in support of these conjectures are presented.

Introduction

Among the principal distinguishing and exciting features of connectionist networks is their capacity for learning. The tendency for these networks to generalize from a limited set of training data to novel stimuli in interesting ways has attracted considerable attention. However, while the connectionist literature contains dozens of examples of generalization, the phenomenon remains poorly understood. Since there is no consensus on the definition of good or appropriate generalization it comes as no surprise that the issue of whether a particular network generalizes "well" or "appropriately" is frequently the subject of debate.

Consider the task of trying to find a "rule" underlying the correlations among some set of data. Assuming the data can be coded quantitatively, this can be cast as a curve fitting task, which depends on choosing an appropriate form for the function (linear, polynomial, Gaussian, etc.). Usually, this choice is motivated by considerations of mathematical simplicity and by an understanding of the domain of the data. An additional factor particular to connectionist models is the compatibility of the rule with the framework of formal neural networks; that is, it must be possible to realize the rule as a network. Typically, the parameters of the system subject to adaptive modification are the weights between nodes.[1] Connectionist networks tend to generalize well, relative to more "brittle" symbolic systems. If for no other reason, this is because, rather than process a set of symbols, which the system may not be equipped to handle, a network computes a vector function, parametrized by the weights, from any vector pattern given as input. Thus, given a novel input, a network always computes an output; this is not a general guarantee of symbol manipulating systems. Of course, the ability to produce an output for any input is not sufficient, in itself, to give appropriate forms of generalization. Other important factors include an appropriate choice of coding for the input and output patterns and careful designing of the network architecture.

This report presents two conjectures regarding the development of internal representations, which are supported by motivating arguments and some simulations on small networks.

Background

Back propagation

Error correction by back propagation of error (backprop or BP), as described by Rumelhart, Hinton, and Williams (1986)[2], has certainly been applied to a wider variety of generalization tasks and with greater success than any other connectionist learning procedure, and has performed well against standard (i.e. non-neural) state-of-the-art techniques, as well. Hence, this report is focused on the BP algorithm; but it should be noted that BP is an example of a more general set of parameter fitting algorithms, to which this work applies. Since the description will involve certain details of the algorithm, a synopsis of BP follows; this will serve to define the notation used throughout the report. The BP algorithm is used to incrementally modify the weights in a feed-forward net, such that, for any of a given set of input patterns $S^1, S^2, \cdots, S^N$, the net will produce the corresponding output pattern T^α. Each step of the modification, or training, process, consists of three stages: selection of a pattern pair α, computing the network's response to s^α, and modifying the weights. The training environment **E** consists of the set of all the pattern pairs S^α, T^α and a probability density across them. In this discussion, only finite, discrete pattern sets will be considered. Let $p(\alpha)$ denote the probability of pattern pair α.

Network architectures will be specified by listing the fan-in set $Fi(i)$ of every non-input unit, i, the set of input units, In, and the set of output units, Out. In addition, it is useful to define the fan-out set, $Fo(i)$ of units which are activated by unit i. Each unit in this discussion is assumed to be a semi-linear unit; that is, each unit computes a linear sum x_i, of its inputs, and passes it through a nonlinear function σ_i, to generate its own activity value r_i:

---

[1] The parameters which are adaptively modified by such a process in a connectionist network can be classified by their order, defined here as the number of nodes directly interacting with the parameter. Thus, for example, a connection strength (i.e., a weight) is of order 2, the bias value used by Rumelhart, Hinton, and McClelland (1986) would be of order 1, a sigma-pi unit (Rumelhart, Hinton, and Williams, 1986) is of order 3, and any global parameter would be of order 0.

[2] It should be noted that the back propagation procedure does not have a single author. Rosenblatt (1962) alluded to a procedure for multilayer networks involving "error propagation". Werbos (1974) devised the algorithm, but his work was not widely recognized for its potential and was thus ignored. Independent developments came over a decade later by Parker (1985) and by Rumelhart, Hinton, and Williams (1986); the latter received much more attention than the prior developments, probably because it was published in the context of a broad approach to cognition.

$$r_i = \sigma_i[x_i \equiv \sum_{j \in Fi(i)} w_{ij} r_j] \qquad [1]$$

The function σ_i, is typically a monotonic function, mapping the real numbers onto a bounded interval; for this reason, it is known as the "squashing" function for unit i. It is a bit unorthodox to use different squashing functions for units within the same network, but such configurations will be considered in this report, for reasons which will become clear.

Each step in the BP algorithm consists of four phases: pattern selection, forward propagation of activity, backward propagation of error, and modification of connection weights. Selection of a pattern pair α is i.i.d. with each step from the environment E. The input portion of the pair, S^α, drives the input units, and activity propagates forward through the network in accordance with Eq.[1]. The procedure for error computation and back-propagation is driven by a gradient descent approach; that is a formula for the change of each weight value is calculated as the negative of the gradient of the error E^α for pattern pair α. Thus, the change induced by pattern pair α, is given by

$$\Delta^\alpha w_{ij} = -\varepsilon \frac{\partial E^\alpha}{\partial w_{ij}} \qquad [2]$$

where ε is a small number (usually less than, or equal to, 0.1), and where the following formula for E^α will be assumed:

$$E^\alpha = \sum_{i \in Out} (T_i^\alpha - r_i^\alpha)^2 \qquad [3]$$

Equations [2] and [3] are sufficient to compute the change for each weight in the network. The back-propagation algorithm is a technique by which an "effective error" δ_i can be computed for each unit, i. This quantity is computed for the output units as the product difference of the desired output T_i^α and the actual response r_i^α, multiplied by the derivative of the squashing function with respect to the current value of x_i^α. For the other units in the network, δ_i is computed by summing the effective errors of its fan-out and multiplying by $\sigma'_i(x_i^\alpha)$:

$$\delta_i^\alpha \equiv \begin{cases} (T_i^\alpha - r_i^\alpha)\, \sigma'_i(x_i^\alpha) & \text{for } i \in Out \\ \sigma'_i(x_i^\alpha) \sum_{j \in Fo(i)} w_{ji}\, \delta_j^\alpha & \text{for } i \notin Out \end{cases} \qquad [4]$$

After the effective errors have been computed, the modification of weights is implemented in analogy to the Perceptron learning rule for single layer networks, in which the output units are activated directly by the input units (Rosenblatt, 1962), and the weights are modified in proportion to the product of the error in the output unit and the activity of the input unit.

$$\Delta^\alpha w_{ij} = \varepsilon\, \delta_i^\alpha\, r_j^\alpha \qquad [5]$$

Generalization

Underlying the notion of generalization is the assumption that the example pairs in the training environment E follow some lawful relation. An important feature of connectionist nets is that they will *always* generalize; that is, having reduced the error across **E** to some proscribed criterion, a lawful relation is defined by the network, which will produce some output pattern for any novel input. The question, then, is whether the relation found by the network matches the relation that exists in the world. An example from the curve-fitting analogy is to find a mathematical relation between the variables x and y, given the example (x,y) pairs (0,0) and (1,1). Even if we restrict the possible solutions to power laws, anything of the form $y=x^p$, for $p>0$, is a correct solution. An example from Boolean algebra would be to complete the truth table $(p,q)\rightarrow r$, given the examples (0,0)→0, (0,1)→1, and (1,0)→1. Generalizing these examples to the estimate

(1,1)→1 completes the OR rule, and the estimate (1,1)→0 gives the EXCLUSIVE OR rule. Both of these examples illustrate the notion that there is not an objectively *correct* solution to a generalization problem; the test is whether the solution *works*.

The feature of BP networks most associated with generalization is an aspect of the architecture known as a *bottleneck*. Let a bottleneck be defined as a pool of units which share identical fan-in and fan-out sets, and for which the number of units in the pool is less than the number of units in the fan-in set. Bottlenecks force distributed representations of the patterns which feed into them from the common fan-in pool.

The use of bottlenecks in connectionist networks began with the description of the encoder architecture by Ackley, Hinton, and Sejnowski (1985), in which a bottleneck of three units was inserted between the input and output layers.[3] The training environment consists of 8 input-output pairs; hence, to succeed, the bottleneck layer must "discover" distinct representations of the 8 input patterns, such that the output units can discriminate among them. Since the discrimination is more efficient if the bottleneck representations are well-separated, the units are often driven to high and low values, and thus the network learns to represent the 8 patterns using a binary code.

Recent work on modifiable architectures has indicated that generalization can be improved by incorporating the number of bottleneck units and/or connections into the cost function minimized by the network (Rumelhart, 1988). In this scheme, the number of units in the bottleneck is minimized concurrently with the error, thus fostering distributed (vis-a-vis local) representations and thereby promoting generalization.

Features in Internal Representations

Hinton (1986) designed a rather elaborate five-layer network to learn kinship relations between the members of a fictitious pair of family trees. His network uses bottlenecks to force distributed representations of orthogonal input patterns, such that the distributed representations enjoy a similarity structure that allow the mapping problem to be solved. In his simulations, the individual units in the bottlenecks come to represent features of the input patterns, which are key to learning the mapping (for example, one unit comes to represent generation and another identifies the branch of the tree). The development of local feature units is intriguing, for it may bear on the issue of local vs. distributed representations. Having developed this similarity structure among the patterns, the network is able to generalize its task to stimulus patterns in novel combinations. For example, if it has been trained that "A is the father of B" and "B is the sister of C", the network can successfully complete the input "who is the father of C?"

Another example of analyzing the features discovered by bottlenecks is the work of Cottrell, Munro, and Zipser (1987), who apply an encoder architecture to the problem of image compression to achieve compression on the order of 1 to 2 bits/pixel using encoders with quantized squashing functions. Analysis of the representations at the bottleneck layer, shows that each unit accounts for a comparable amount of error. On inspection, each bottleneck unit seems be be tuned to a particular orientation and a particular spatial frequency; curiously, neurons in mammalian visual cortex are sensitive to these features as well.

Representations in a three layer net: two conjectures

While the above examples indicate a tendency for the units in bottlenecks to pick out identifiable features of the input patterns, this does not universally occur. In a BP approach to mapping locative expressions (eg. "boat in water") to spatial relationships (eg. first noun on top of and supported by second noun), it was found (Cosic & Munro, 1988), that the mapping was successfully learned, and it generalized well. Since Herskovits (1986) has demonstrated that this mapping depends on certain features of the nouns' referents, bottlenecks were included in the network immediately following the noun input patterns, which were orthogonal. It was expected, based on Hinton's (1986) family trees result, that units would develop to represent these features if the mapping were successfully learned. However, this was not the

---

[3] Ackley, Hinton, and Sejnowski designed the encoder for, and demonstrated it using, the Boltzmann machine. A stochastic network with a different learning algorithm. Their work predated Rumelhart, Hinton, and Williams (1986) description of BP, in which they apply BP to the encoder problem successfully.

case. While nouns whose referents share similar physical features have similar *representations* at the bottleneck, these features do not map onto individual *units.*

Since the absence of microfeatures does not seem to hinder performance, it might be argued that they are inconsequential. Some indirect evidence to the contrary is provided in the next section.

The Squashing Function

Consider the squashing function. A common choice is the logistic function L(x):

$$L(x) \equiv \frac{1}{1+e^{-x}} \qquad [6]$$

This function maps the real numbers onto the range [0,1]. This may have important ramifications for representations in bottlenecks using this squashing function, since the components are restricted to positive values. Thus, the representation vectors must all lie in the same orthant (the analogy of a 2-D quadrant in multi-dimensional space). In contrast, a linear transformation of L(x), N(x) : 2L(x) - 1, has [-1,1] as its range. Thus the representation vectors of an n-unit bottleneck are distributed in a space which is more "open" by a factor of 2^n; i.e. the number of possible vector directions is multiplied by a factor of 2^n.

This constraint on the set of representations leads to the following two conjectures, for two squashing functions, σ and σ^*, which are arbitrary, subject to the conditions that they are continuous, differentiable, and $\sigma:\mathbf{R}\rightarrow[0,1]$ where $\sigma^*\equiv 2\sigma-1$. Note that they need not be monotonic and that $\sigma^*:\mathbf{R}\rightarrow[-1,1]$.

> *Conjecture 1:* Networks with bottlenecks using squashing function σ should adapt more slowly with respect to a fixed criterion than identical networks using σ^*.
>
> *Motivation:* Since the representations distributed over the region $[-1,1]^n$ are more "open" by a factor of 2^n than those distributed over the region $[0,1]^n$; i.e. the number of possible vector directions is multiplied by a factor of 2^n. Thus, representations in bottlenecks should be more readily separated if σ rather than σ^* is used. See Figure 1.
>
> *Conjecture 2:* Units in bottlenecks using σ are expected to show a tendency toward developing local feature units relative to those using σ^*.
>
> *Motivation:* Observe that orthogonal vectors confined to the positive orthant must have zero components and hence lie on a boundary of the region, and that if n vectors are chosen in the positive orthant of n-space, they must lie along the axes. Under the assumption that BP seeks to orthogonalize distinguishing features of the environment in bottlenecks, these features will show a tendency to lie near the coordinate axes of the representation space, if the function σ is used; these axes correspond to individual units.

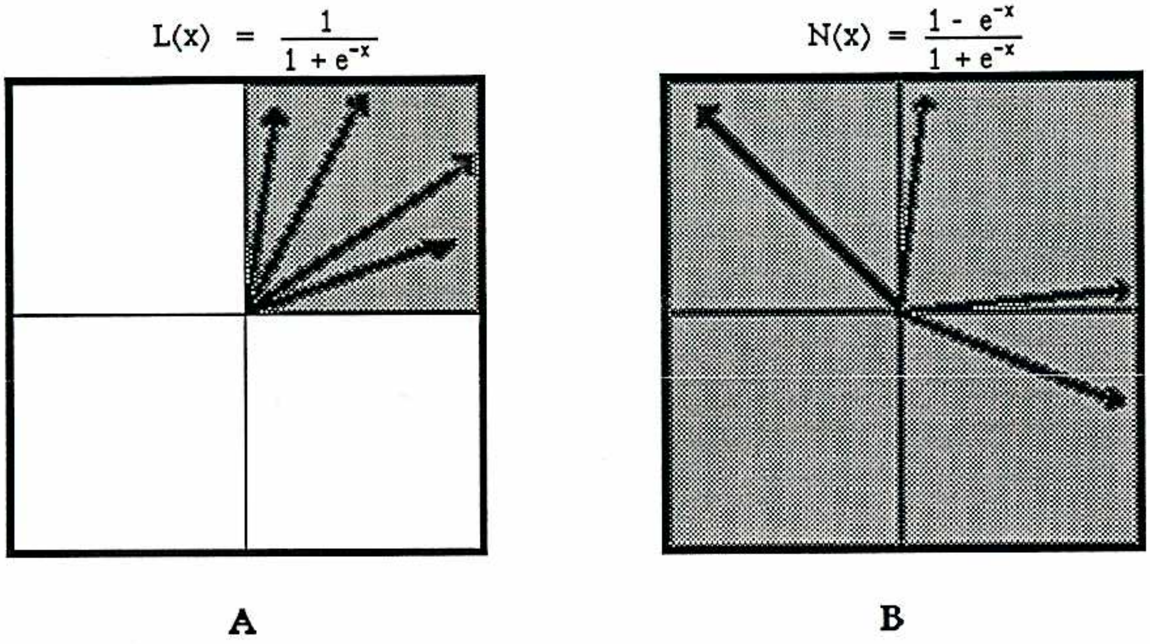
$L(x) = \frac{1}{1 + e^{-x}}$
$N(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$
A
B

Figure 1. Allowed vectors in a 2-D representation space. **A.** The allowed region for vectors using L(x) is shaded with some sample vectors. **B.** The allowed region for vectors using N(x) is shaded with some sample vectors. Note that the vectors in A are more similar than those in B, in that the average angle between them is smaller, and hence more resistant to discrimination.

The Bias Level

A bias parameter is attached to every node in most BP applications. Formally, the bias is equivalent to a weight from a unit that is always on, with strength β. So, if the bias for unit i is defined to be θ_i, we have the following:

$$x_i^\alpha = \beta\theta_i + \sum_j w_{ij}\, r_i^\alpha \qquad [7a]$$

$$\Delta^\alpha \theta_i = \varepsilon\, \beta\, \delta_i^\alpha \qquad [7b]$$

The bias unit strength, β, is constant through the network and over time, for a particular simulation. Other models using the bias parameters θ_i have always implicitly assumed β=1. It is introduced here as a parameter of the model to be varied across different simulations (i.e., like ε). Hence, β acts as an constant input component to the fan-out pool of a bottleneck and, as such, plays a role in determining the similarity structure of the representations at a bottleneck.

Consider, as in Figure 1, a bottleneck consisting of just two units, unit m and unit n. Unit i in the fan-out pool, is now activated by a three dimensional pattern (r_m, r_n, β) that is weighted by the corresponding parameters $(w_{im}, w_{in}, \theta_i)$. The domain of possible patterns in the 3-space defined by units m, n, and the bias, is now restricted to the plane $z=\beta$, bounded by $a<r_m<b$ and $a<r_m<b$, where a and b are respectively the lower and upper bounds of the squashing function. Examples of allowed regions for various a,b,β, are shown in Figure 2.

The similarity matrix of the representation vectors is thus related to the values of a,b,andβ. Since all regions of this kind are related by linear scale transformations, the choice of region will not have an effect on certain aspects of the space. For example, two sets of vectors are linearly separable either for all combinations of these parameters or for none of them (of course, it is assumed that $a<b$). However, certain properties of the solution may be subject to influence by these parameters, as indicated by the two conjectures.

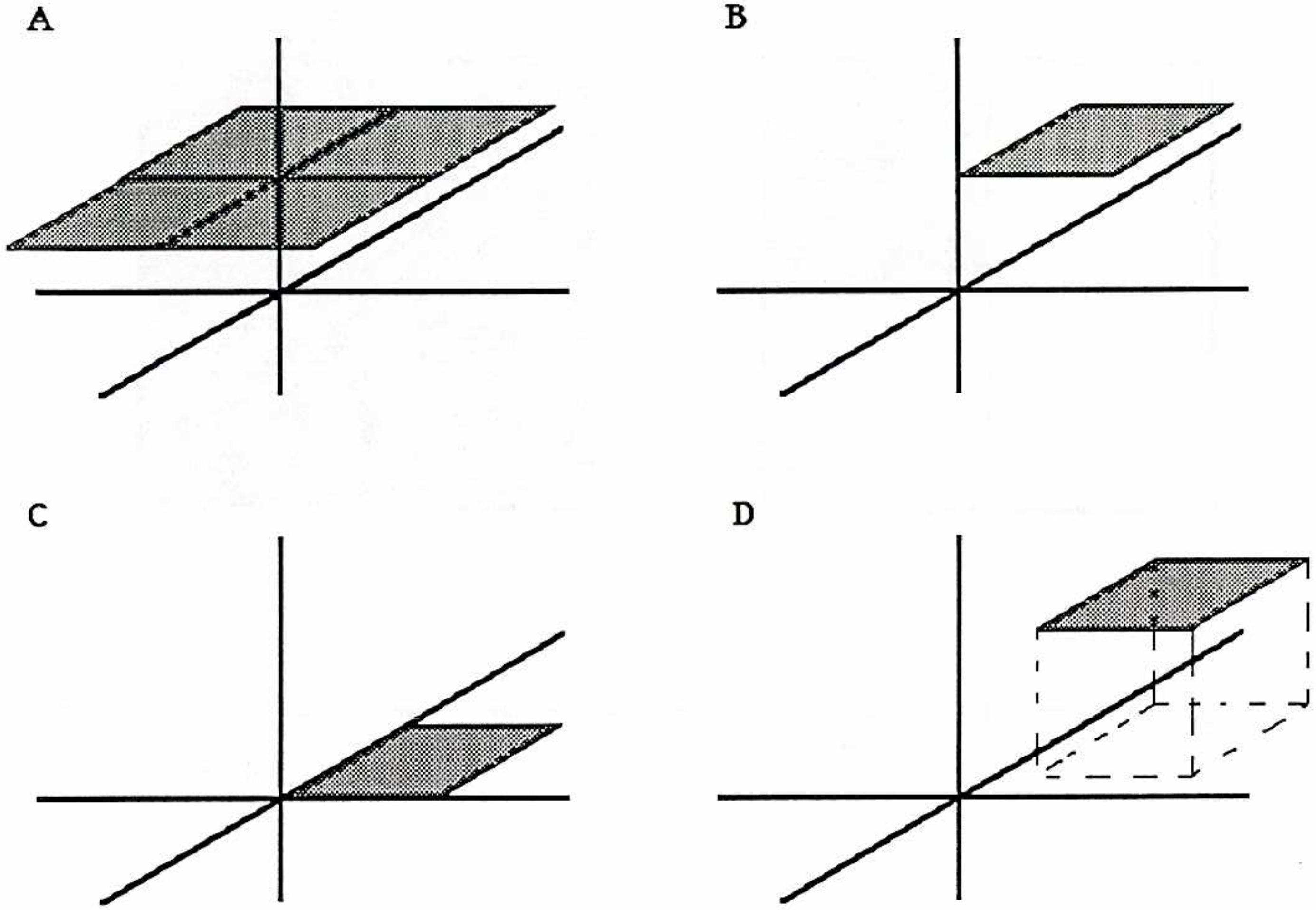

Figure 2. Regions of allowed pattern vectors. The allowed pattern vectors must reside in a restricted region of (n+1)-space where n is the number of components in the representation. Here, the vertical axis is the bias dimension, β. The following parameter values are used in the diagrams: **A.** $\beta=1$, $a=-1$, $b=1$. **B.** $\beta=1$, $a=0$, $b=1$. **C.** $\beta=0$, $a=0$, $b=1$. **D.** $\beta=1$, $a=0.5$, $b=1.5$.

Simulations

Some preliminary simulations have been run, which provide some support for the conjectures. The networks are strictly layered, with a bottleneck of just two units inserted between the input pool and the output pool. In each case, the input values and the training values are set at 0 and 1, and so the function L(x) is used as the squashing function by the output units.[4] However both L(x) and N(x), as well as a variety of bias values are used for the bottleneck units of each problem. Figure 3 shows the hidden unit representations formed by a 2-2-2 encoder. The final states show that, for the simulations using N(x), as the bias strength increases from 0 to 1, the angle between the representation vectors goes from perpendicular to antiparallel. In three dimensions (counting the bias component), the angle remains roughly perpendicular. Figure 3D shows the development of representations using L(x); note that the final states are near the axes.

---

[4] Following the suggestion of Plaut, Nowlan, and Hinton (1986), the training values are sometimes set at values close to, but inside of, the bounds of the squashing function.

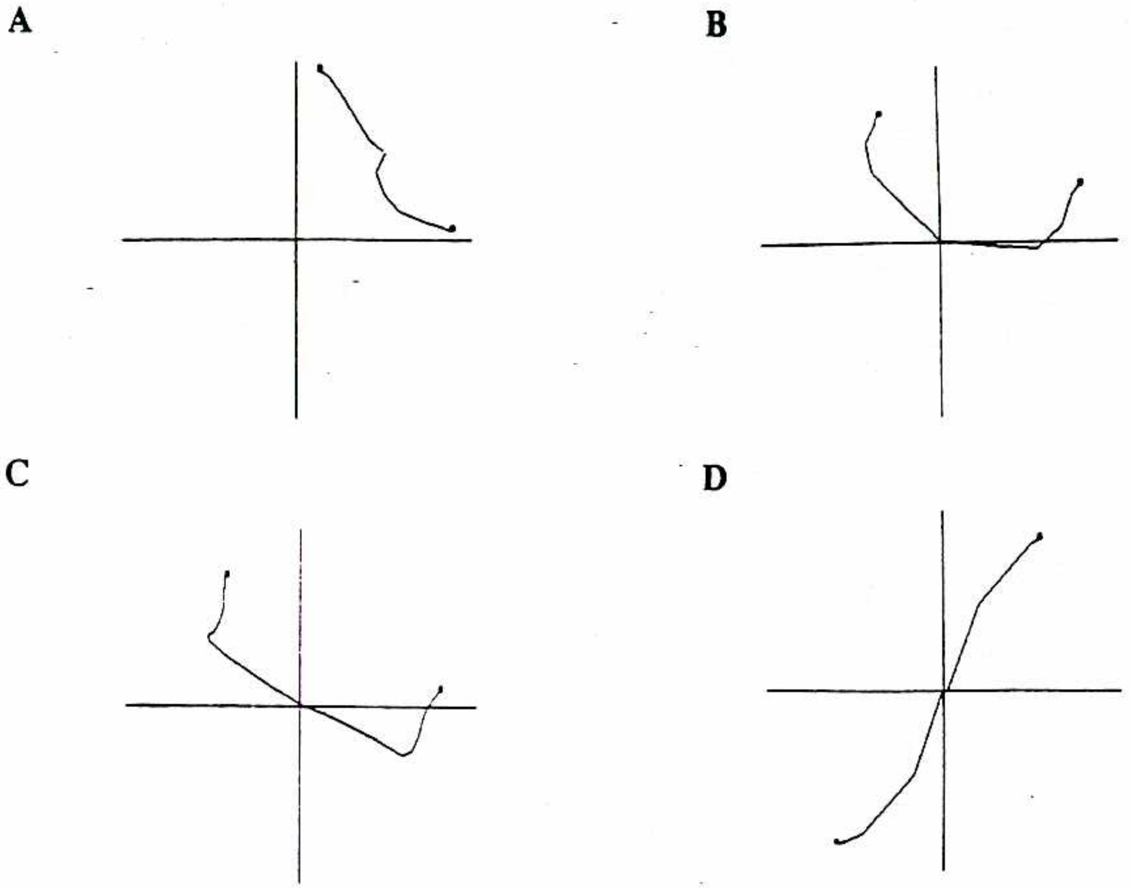

Figure 3. Bottleneck representations developed by a 2-2-2 encoder. This figure shows the evolution of representations by BP for L(x) and for various values of the bias using N(x). The axes correspond to activities of the two units in the bottleneck. The lines are the activity trajectories over time, the small squares are the final states. A. L(x), β=1. B. N(x), β=0.1. C. N(x), β=0.2. D. N(x), β=1.

In Figure 4, the effect of the squashing function is illustrated. The evolution of representations from three patterns in a 3-2-3 encoder is traced. In both cases, the patterns are well separated. Note that the representations that develop under the L(x) function lie at the corners of the quadrant, whereas a different configuration evolves under N(x). This demonstrates that the representations formed under these two functions are *qualitatively* distinct; that is, the difference in the arrangements does not differ by a simple scale factor (even though the transfer functions are linearly related).

Finally, results from two 4-3-6 networks are presented below. The input/output environment is shown in Table 1. Note that the input patterns are orthogonal, but that output patterns 1 and 2 are quite similar, as are patterns 3 and 4, and that the two groups are mutually perpendicular. The results of simulations using N(x) and L(x) at the bottleneck are compiled, in Tables 2 and 3 respectively, after the error has been reduced to a low level.

Table 1. Environment for the 4-3-6 Network Simulations

α	Input Pattern				Output Pattern					
1	1	0	0	0	1.0	1.0	0.5	0.0	0.0	0.0
2	0	1	0	0	0.5	1.0	1.0	0.0	0.0	0.0
3	0	0	1	0	0.0	0.0	0.0	1.0	1.0	0.5
4	0	0	0	1	0.0	0.0	0.0	0.5	1.0	1.0

Table 2. Results from the 4-3-6 network after 1600 iterations using N(x)

Input	Bottleneck			Output						Error
1	0.194	0.910	-0.348	0.957	0.979	0.505	0.017	0.021	0.024	0.00
2	0.851	0.044	-0.893	0.503	0.977	0.956	0.025	0.023	0.020	0.00
3	0.275	-0.863	0.882	0.012	0.020	0.029	0.961	0.980	0.504	0.00
4	-0.847	-0.868	0.108	0.015	0.019	0.019	0.503	0.981	0.964	0.00
Total error =										0.01

Table 3. Results from the 4-3-6 network after 2000 iterations using L(x)

Input	Bottleneck			Output						Error
1	0.909	0.042	0.106	0.917	0.976	0.525	0.038	0.024	0.009	0.01
2	0.953	0.782	0.031	0.520	0.957	0.878	0.005	0.043	0.070	0.02
3	0.046	0.148	0.974	0.054	0.041	0.009	0.914	0.959	0.509	0.01
4	0.111	0.954	0.933	0.004	0.019	0.060	0.528	0.981	0.920	0.01
Total error =										0.06

Both conjectures are supported by the simulations. Conjecture 1 is supported by the observation that the total error is higher after more iterations in the L(x) simulation than in the N(x) simulation (both started with the same weights and parameter settings). Conjecture 2 is supported by analyzing the hidden unit representations. Note that unit 1 and unit 3 in the bottleneck have responses near the ceiling and the floor of L(x), such that unit 1 has become a "feature detector" for the sibling patterns 1 and 2, and unit 3 has the converse role with the other siblings, patterns 3 and 4. Unit 2 is used to discriminate between patterns within a pair. The bottleneck units in the N(x) simulation don't exhibit this quality of flagging specific features so clearly, although units 2 and 3 seem to show a tendency in this direction.

Future directions

Representations

An analysis of several network architectures and pattern environments is planned, specifically, but not exclusively including evaluation of the representations developed under various squashing functions and parameters (eg. the bias strength). While the verification of Conjecture 2 may seem inconsequential to the function of connectionist nets, it would have the following ramifications. First, it would lead to a more complete understanding of the learning process in multilayer networks of nonlinear units. Second, a recipe for producing feature detecting units would be a useful tool for research and development in neural networks. Third, further substantiation of Conjecture 2 (either analytical or empirical) may integrate two phenomena from neurobiology: (a) Every neuron acts in a purely inhibitory or excitatory fashion -- that is representations must lie within a particular orthant, and (b) neurons with identifiable trigger features abound, at least in sensory cortex.

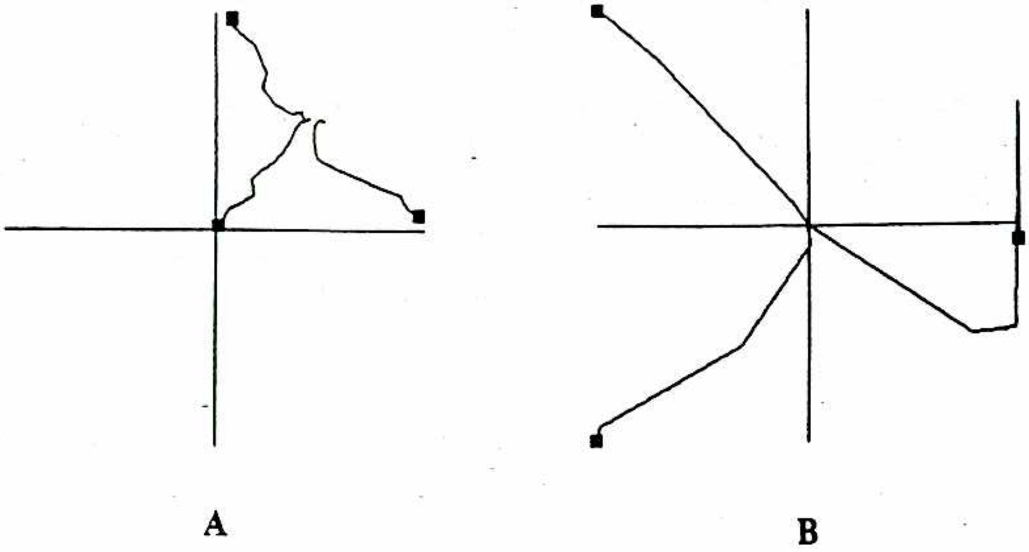

Figure 4. Bottleneck representations developed by a 3-2-3 encoder. This figure shows the evolution of representations by BP for A. L(x) and B. N(x). See text for description.

Generalization

Various factors, such as weight decay (Plaut, Nowlan, and Hinton, 1986) have been shown to have an effect on generalization. Recent experiments using minimization of the bottleneck have also shown promise (Rumelhart, 1988). Another direction for future work will be to establish a relation between generalization and some factors which have not yet been analyzed in this context, such as bias strength and bounds on the squashing function. Analysis of these particular parameters is motivated by the arguments and preliminary studies presented in support of the two conjectures. Since generalization seems to be strongly related to internal representations, it is difficult to imagine studying one without the other.

Another aspect of learning, which may bear on the issue of generalization, is the order of pattern presentation. In a typical simulation, patterns are presented to networks randomly, independent of the previously presented pattern (excluding networks trained on sequences). This may lead toward the development of a network that could influence its environment, such that it could dynamically alter the probability distribution for the presentations. A simple example would be to allow the network to have a stimulus repeated under certain conditions (if it gave a particularly high error, for example). Networks with such an ability to interact with their environment may exhibit greater capacities for learning and generalization.

Extension to larger networks

The complexity and scale of the networks described above is minimal. While this greatly facilitates the detailed analysis, it begs the question of whether the results will be relevant for networks designed to handle real-world problems. Thus, work is also planned toward extension of the results of the analyses of representation and generalization to networks of increasing complexity. The first candidate is to work with small, and then with larger versions of Hinton's (1986) family trees network. Other candidates include Sejnowski and Rosenberg's (1986) NetTalk, Munro's (1987) scalar reinforcement network, Cottrell, Munro, and Zipser's (1987) image compression network, and Cosic and Munro's (1988) locative preposition network.

With greater complexity come opportunities for exploring more subtle and varied factors of the simulation. For example, several networks have multiple bottlenecks; it may prove fruitful to investigate simultaneously using different parameters on different bottlenecks within a network.

Networks with several intermediate layers (eg. Linsker, 1986) are expected to show complex courses of learning. An important open issue for examination is to consider the order in which these layers organize. In supervised training paradigms such as BP, the environment acts on the network from both ends. Hence, the feed forward activations are not stable at the upper layers until the lower representations settle down, but the feed back error signals do not stabilize until the upper weights reach equilibrium. A hint to this chicken and egg dilemma comes from an analysis of developmental stages in neurons of the primate visual system (Harwerth, Smith, Crawford, Duncan, and von Noorden, 1986), who give evidence that stages closer to the input stabilize earlier. A theoretical explanation for this is offered by Munro (1986).

References

Ackley, D., Hinton, G., and Sejnowski, T. (1985) A learning algorithm for Boltzman machines. *Cognitive Science* 9: 147-169.

Cosic, C. and P. Munro (1988) Learning to represent and understand locative prepositional phrases. *Proc. Tenth Ann. Conf. Cognitive Science Society* pp. 257-262.

Cottrell, G. W., P. W. Munro, and D. Zipser (1987) Image compression by back propagation: An example of extensional programming. *Proc. Ninth Ann. Conf. Cognitive Science Society* pp.461-473.

Harwerth, R. S., Smith, E. L., Duncan, G. C., Crawford, M. L. J., and von Noorden, G. K. (1986) Multiple sensitive periods in the development of the primate visual system. *Science* **232:** 235-238.

Herskovits, A. (1986) *Language and Spatial Cognition.* Cambridge: Cambridge University Press.

Hinton, G. (1986) Learning distributed representations of concepts. *Proc. Eighth Ann. Conf. Cognitive Science Society* pp. 1-12.

Linsker, R. (1986) From basic network principles to neural architecture: Emergence of orientation columns. *Proc. Nat. Acad. Sci.* **83:** 8390

Munro, P. W. (1986) State-dependent factors influencing neural plasticity: a partial account of the critical period. In: *Parallel Distributed Processing: Explorations in the microstructure of cognition.* J. L. McClelland and D. E. Rumelhart, eds. Cambridge MA: MIT Press.

Munro, P. W. (1987) A dual back-propagation scheme for scalar reward learning. *Ninth Ann. Conf. of the Cog. Sci. Soc. Proc.* pp. 165-176.

Parker, D. B. (1985) *Learning-logic.* (Report TR-47) Cambridge MA: MIT Center for Research in Computational Economics and Management Science.

Plaut, D., Nowlan, S., and Hinton, G. (1986) Experiments on learning by back-propagation. (Report CMU-CS-86-126) Pittsburgh PA: CMU Department of Computer Science.

Rosenberg, C. R. and Sejnowski, T. J. (1987) Parallel networks that learn to pronounce English text. *Complex Systems* **1:**145-168.

Rosenblatt, F. (1962) *Principles of Neurodynamics.* New York: Spartan.

Rumelhart, D. E. (1988) Unpublished remarks at his plenary talk of the ICNN meeting. San Diego, CA. July, 1988.

Rumelhart, D. E. , Hinton, G., and McClelland, J. (1986) A general framework for parallel distributed processing. In: *Parallel Distributed Processing: Explorations in the microstructure of cognition.* J. L. McClelland and D. E. Rumelhart, eds. Cambridge MA: MIT Press.

Rumelhart, D. E., Hinton, G., and Williams, R. (1986) Learning internal representations by error propagation. In: *Parallel Distributed Processing: Explorations in the microstructure of cognition.* J. L. McClelland and D. E. Rumelhart, eds. Cambridge MA: MIT Press.

Werbos, P. (1974) Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD Thesis. Harvard University, Cambridge MA.