In: *Backpropagation: Theory, Architectures and Applications*, Chauvin and D. Rumelhart (eds),

Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pages 313-349.

Spatial Coherence as an Internal Teacher for a Neural Network

Suzanna Becker and **Geoffrey E. Hinton**

Department of Psychology
McMaster University
1280 Main Street West
Hamilton, Ontario
Canada, L8S 4K1

Department of Computer Science
University of Toronto
6 King's College Road
Toronto, Ontario
Canada, M5S 1A4

Contents

Abstract

Supervised learning procedures for neural networks have recently met with considerable success in learning difficult mappings. So far, however, they have been limited by their poor scaling behaviour, particularly for networks with many hidden layers. A promising alternative is to develop unsupervised learning algorithms by defining objective functions that characterize the quality of an internal representation without requiring knowledge of the desired outputs of the system. Our major goal is to build self-organizing network modules which capture important regularities in the environment in a simple form. A layered hierarchy of such modules should be able to learn in a time roughly linear in the number of layers. We propose that a good objective for perceptual learning is to extract higher-order features that exhibit simple coherence across time or space. This can be done by transforming the input representation into an underlying representation in which the mutual information between adjacent patches of the input can be expressed in a simple way. We have applied this basic idea to develop several interesting learning algorithms for discovering spatially coherent features in images. Our simulations show that a network can discover depth of surfaces when trained on binary random dot stereograms with discrete global shifts, as well as on real-valued stereograms of surfaces with continuously varying disparities. Once a module of depth-tuned units has developed, we show that units in a higher layer can discover a simple form of surface interpolation of curved surfaces, by learning to predict the depth of one image region based on depth measurements in surrounding regions.

1 Introduction

The mammalian perceptual system can recognize complex stimuli with remarkable speed and robustness. So far, the performance of computational models of perceptual skills such as the recognition of continuous speech and complex scenes has fallen far short of the performance level of humans on these tasks. How does the brain achieve such remarkable pattern recognition abilities? A partial explanation may be that the brain undergoes substantial self-organization as the young animal gradually learns to find regularities in the complex patterns of stimuli which it encounters. By examining the ways in which the brain employs self-organizing principles, we can gain some insight as to how we should design efficient, and biologically plausible computational models of learning.

1.1 Evidence of activity-driven self-organization in the brain

There is substantial evidence that brain development and perceptual functioning are markedly affected by exposure to structured environmental input. In this section, we briefly review some of the neurophysiological evidence for these claims. For a more extensive review, see (Miller, 1990).

Much of the evidence comes from single cell recordings in the primary visual cortex of cats (Area 17) and more recently, cell recordings in primate visual and somato-sensory systems. In several mammals, visual cortical neurons have been found to be tuned to a variety of stimulus features such as orientation (i.e., some cells respond maximally to edges or bars of particular orientations), spatial-frequency, velocity and direction of movement, and binocular disparity. The fine-tuning of the visual system to some of these features occurs primarily during certain "sensitive periods of development" early in life, and can be disrupted if an animal is reared in an impoverished environment.

When kittens are raised in environments containing only vertical or horizontal contours, cortical cell populations exhibit orientation selectivities which are skewed in favor of those particular contours to which the animal was exposed(Blakemore and van Sluyters, 1975; Blakemore and Cooper, 1970; Hirsch and Spinelli, 1970). However, since orientation tuning is present in visually inexperienced kittens (Hubel and Wiesel, 1963), it is unclear whether the effect of distorted environmental input is to actually alter the orientation selective response of single cells (e.g., via a competitive activity-dependent process), or to simply cause deterioration of these cells as a consequence of their disuse(Miller, 1990). Spatial-frequency tuning of feline cortical cells is also influenced by early visual experience: while no visual input is necessary for normal development of spatial-frequency selectivity during the first three weeks, the subsequent fine-tuning of frequency sensitivity which normally occurs in weeks 4-8 is blocked if the cat is deprived of patterned input (Derrington, 1984).

Another widely studied developmental phenomenon in the cat's visual cortex is the appearance of ocular dominance columns (ODCs); after normal binocular visual experience, cortical cells which receive input from both eyes eventually segregate into alternating eye-specific patches. This effect can be strongly influenced by environmental factors: when a cat is subjected to monocular deprivation, particularly during the second postnatal month, cortical cells become almost exclusively and irrecoverably responsive to input from the exposed eye (Wiesel and Hubel, 1965; Olson and Freeman, 1980). However, if *both* eyes are deprived of exposure to patterned input (e.g., in dark-reared and eyelid-sutured kittens), there is evidence that ocular dominance segregation may still occur to a some degree (Stryker and Harris, 1986). It is possible that spontaneously generated, random input from retinal cells may be a factor; when retinal ganglion cell activity is completely blocked with a drug called tetrodotoxin during the period in which ocular segregation normally occurs in the cortex, ODCs fail to develop (Stryker and Harris, 1986).

There is similar evidence of activity-dependent development in primate perceptual systems. For example, monkeys with optically induced strabismus (a misalignment of the two eyes, either convergent or divergent) early in life, followed by three years of normal binocular experience, have a marked reduction in the number of cortical binocular cells (in both areas V1 and V2) and are behaviourally stereoblind (Crawford et al., 1984). Interestingly, children with esotropia (convergent strabismus, a condition commonly referred to as "cross-eyed") in infancy, which is surgically corrected between 10-13 years of age, perform identically to the strabismic monkeys on stereopsis tasks (i.e., they are clinically stereoblind), suggesting the possibility of a similar critical period in humans for the development of binocular vision (Crawford et al., 1983).

Cortical reorganization occurs not only during sensitive periods early in development, but continues through the lifetime of an animal, as a result of changing environmental demands. For example, Merzenich and his associates (1987) have studied plasticity in the somato-sensory cortex of adult owl monkeys (for a review, see (Merzenich et al., 1988)); neurons in this region of the brain respond to tactile input from local areas of the skin surface, forming roughly a topographic map of the body. Merzenich's group has extensively studied cells whose receptive fields correspond to regions of skin surface on the hand (in Area 3B); these receptive fields change drastically when the spatio-temporal correlations of tactile stimulation are experimentally varied.

It is possible that cortical cells employ some very general, sensory modality-independent organizing principles in developing their characteristic responses to patterned input. Startling evidence for this possibility comes from an unusual study in ferrets (Sur, Garraghty and Roe, 1988). Nerve fibers from the primary visual pathway were artificially redirected into the auditory cortex, and these visual inputs formed synaptic contacts with cells in the auditory cortex. After the ferrets were reared to adulthood, the majority of "auditory" cortical cells tested had become visually driven, some having center-surround response characteristics similar to cells in the visual cortex.

The ability to develop a set of environmentally tuned feature detectors is of clear adaptive advantage. The organism need not have a completely genetically predetermined perceptual system; rather, the primitive organization laid out by the genetic blueprint can be fine-tuned after birth, as the statistics of the environment, and even the physical properties of the organism's own sensors (e.g., the distance between the two eyes, two ears etc.) change over time. Once an environmentally tuned set of feature detectors has been learned, important features of the world such as local variations in position and depth can be quickly extracted by individual cells or cell populations in parallel. The response of these cells can then serve as a preprocessed input to higher layers, which can in turn self-organize to form still more complex representations.

1.2 Computational models of learning

Motivated by the evidence of activity-dependent self-organization in the brain, described above, and by the poor scaling behaviour of supervised learning procedures, our goal is to discover ways in which artificial systems can perform unsupervised learning. By discovering good solutions to this problem in simulated networks of neuron-like processing elements, we hope to cast light on how the brain may solve the same problem.

Much of the current research on computational models of learning in neural networks focuses on learning algorithms in which the network learns from an external teacher. The goal of the network is to learn some prespecified mapping, so that the output of the network matches the teaching signal on each case. Supervised learning algorithms have met with considerable success in solving some difficult tasks, such as sub-problems of speech recognition (e.g. (Lang and Hinton, 1988; Watrous and Shastri, 1987)) and image understanding (e.g. the recognition of hand-drawn digits (Le Cun et al., 1989). However, these algorithms have so far been limited by their poor scaling behaviour, particularly when applied to full-scale real world problems (like continuous speech recognition) which require large networks with many hidden layers. Additionally, they tend to find problem-specific representations which may not carry over well to new situations.

One solution to these problems may be to make use of unsupervised learning, as the brain apparently does: rather than trying to solve difficult problems in one stage by training a large multi-layer network, we can subdivide the problem. First we build adaptive preprocessing modules that can capture some of the interesting features in the environment, and form representations of a simpler form than the raw, unprocessed input. Modules can then be assembled hierarchically to extract features of progressively higher order. Finally, once the hierarchy of unsupervised modules has extracted the important underlying causes of the perceptual inputs, we add supervised modules on top which can quickly learn to associate responses with the causes of the perceptual input rather than the input itself.

1.3 Related work on unsupervised learning

Various approaches have been taken to modeling unsupervised learning processes. In this section, we briefly review examples of two major approaches. We then discuss what type of data distributions each is best at modeling, and finally, what kind of representation should be extracted to be useful for further stages of learning.

1.3.1 Clustering algorithms

An example of a fairly simple, yet impressive application of the idea of using an unsupervised preprocessing layer to speed up subsequent supervised learning is the recent work of Moody and Darken (1989). They trained an adaptive preprocessing layer of Gaussian units by a standard k-means clustering algorithm; an additional layer of units was then trained by a supervised learning procedure (mean squared error minimization) to solve several difficult problems: phoneme recognition and chaotic time series prediction. They report that this hybrid algorithm results in about two orders of magnitude speedup in the learning time compared to pure supervised learning with back-propagation. The layer of Gaussian units speeds learning for two reasons: Firstly, two very different input vectors will tend to activate non-overlapping sets of Gaussian units, so there will be no interference (generalization) between these two training cases. Secondly, the incoming weights of Gaussian units do not depend on the outgoing weights so there is modularity of learning between these two layers of weights.

Many algorithms have been developed for unsupervised learning in artificial neural networks. Many of these are variants of statistical clustering algorithms and are generally referred to as competitive learning procedures (e.g. (Grossberg, 1987; Von der Malsburg, 1973; Rumelhart and Zipser, 1986; Kohonen, 1982; Fukushima, 1975)). Clustering algorithms such as k-means or competitive learning are useful when the goal is to find a limited number of prototypes in high-dimensional data. Each competitive unit or module comes to encode the centre of a cluster of data points in a high-dimensional space, and considerable dimensionality reduction can be achieved, provided that the data points all lie on or near a lower-dimensional surface (Steve Nowlan, *personal communication*). With a more elaborate architecture employing recurrent connections, Grossberg's version of competitive learning (Grossberg, 1987; Carpenter and Grossberg, 1987) continuously adapts to new inputs, while maintaining relatively stable representations, and with some parameter-tuning, gives one control over the plasticity-stability tradeoff. Recently, a "soft competitive learning" method has been proposed (Nowlan, 1990); rather than only allowing the winner to adapt, each unit can adapt its weights for every input case, so that as a group the units form a good multi-modal model of the underlying distribution.

1.3.2 Principal components analysis

Another major approach to unsupervised learning is based on the objective of representing the principal components of the input distribution.[1] Several researchers have developed learning algorithms which accomplish part or all of this task. Oja (1982) showed that a normalized version of the simple Hebbian learning rule allows a unit to extract the first principal component of the input. Extending this idea, Oja's Subspace learning algorithm (Oja, 1989) finds a set of k weight vectors which span the same subspace as the first k principal components. The same is true of the weights learned by an auto-encoder (Baldi and Hornik, chapter ??, this volume; Baldi and Hornik, 1989; Bourlard and Kamp, 1988). Sanger's Generalized Hebbian Algorithm (Sanger, 1989a; Sanger, 1989b) allows a group of linear units to learn a full principal components decomposition of the input distribution by combining Hebbian learning with Gram-Schmidt orthogonalization. Finally, Foldiak (1990) has described a related, but more biologically plausible algorithm that uses anti-Hebbian learning.

A principal components representation has several nice optimality properties. First, it can be shown to minimize the mean squared error in optimal linear reconstruction of the inputs (for a

[1] i.e. the eigenvectors of the input correlation matrix.

detailed analysis, see (Sanger, 1989c)). Second, Linsker (1988) has pointed out that under certain conditions the principal component analyzing unit resulting from a Hebbian learning rule also transmits maximal information about its inputs. Hence this approach to unsupervised learning is useful if the main objective is to achieve maximal data compression, while retaining as much information as possible about the data. On the other hand, even Linsker (1989) has questioned the relevancy of accurate input reconstruction as a goal for biological perceptual systems.

1.3.3 How can we best model the input distribution?

Ideally we would like to be able to capture all the "interesting features" in the data. However, it is unlikely that a single unsupervised learning method can do this in reasonable time for any arbitrary input distribution, without making some *a priori* assumptions about the kinds of structure in the environment. Each of the approaches we have mentioned can be expected to perform well on certain distributions, and poorly on others.

Algorithms related to Principal Components Analysis (PCA) learn a set of linear orthogonal projections in the directions of principal variation in the input distribution, or a rotated subspace of these directions. In some cases, the principal components decomposition may coincide with features of interest in the input. Oja (1989) suggests that subspace methods are particularly useful for representing and classifying patterns such as spectra and histograms. Linsker (1986a, 1986b, 1986c) shows how interesting feature detectors similar to those found in early visual cortex can develop via a Hebbian learning mechanism in a linear network operating on spatially localized patches of random inputs. Miller, Keller and Stryker (1989) have performed similar simulations with units having binocular inputs and local lateral excitatory connections; using a Hebbian learning rule with decay in this case leads to the formation of ocular dominance columns.

For arbitrary input distributions, there is no guarantee that a subspace method or PCA will capture the interesting structure. PCA would be expected to represent poorly parameters which vary in a highly nonlinear manner, and may in fact, depending on the distribution, obscure clusters in the data (i.e. if there are too many isotropically distributed clusters (Huber, 1985)).

In situations where PCA fails to find any interesting structure in the data, competitive learning schemes may in some cases be advantageous, since they attempt to represent explicitly the locations of clusters. Each cluster may implicitly come to represent conjunctions of interesting features, which can be interpreted as describing prototypical class members. On the other hand, the effectiveness of such learning schemes is likely to be very dependent on the number and distribution of clusters in the data, hence, the architecture must be carefully chosen to suit the input distribution. Further, most competitive learning algorithms based on a hard "winner-take-all" strategy are very sensitive to the initial conditions (i.e. initial random weights), and the order of presentation of input patterns.

1.3.4 What kind of representation is best for later learning?

In designing unsupervised learning algorithms, in addition to the question of what type of representation best models the data, we must address a related question: what type of representation will be most convenient as input for further learning?

PCA has the nice property of finding a set of *uncorrelated* projections. This will typically be very helpful for a method such as back-propagation learning by steepest descent. One thing that makes supervised back-propagation learning slow is that there may be interacting effects of different weights on the error. Because of these interactions, convergence can be expected to be very slow, even with simple local accelerating procedures such as momentum (Plaut, Nowlan and Hinton, 1986) or delta-bar-delta (which employs an adaptive learning rate for each weight based on gradient information) (Jacobs, 1987). If, on the other hand, the input representation uses uncorrelated components, then the Hessian of the error function is diagonal, so simple acceleration methods will permit a considerable speedup, by scaling the learning rates appropriately along each weight axis independently. Of course, this analysis ignores the non-linearities of a multi-layer back-propagation network, but it may still provide a useful heuristic for designing good input representations even in the nonlinear case.

If later learning requires interpolating a smooth function from points in the input distribution, then any reasonable set of radial basis functions will make a good set of interpolation points (Moody and Darken, 1989; Renals and Rohwer, 1989; Poggio, 1989). However, in some situations there are disadvantages to this type of representation. We may be collapsing information about underlying features of the data within each cluster; we are thereby scattering essential information about the continuity of individual features, and about combinations of these features, across many clusters. Suppose, for example, that there is a weak correlation between the desired output and the first component of the input vector. Since the correlation is weak, a large sample is required to distinguish between the correlation and sampling error. To get a large sample it may be necessary to combine data points that differ considerably on the values of *other* components. In a system that uses narrowly-tuned radial basis functions, only a few of the training cases will activate each RBF, and so the information about the weak correlation will be spread over many different RBF's and it will be hard to detect the correlation. RBF's make it easier to detect correlations between the desired output and very high order combinations of input values, but harder to detect some of the low order correlations that are easily detected by systems that use a less local representation.

While clustering and PCA-related algorithms may be useful forms of preprocessing for achieving data compression, noise reduction, and smoothing, we would like to take unsupervised learning much further. Our major goal is to build self-organizing network modules which capture important regularities of the environment in a simple form suitable for further perceptual processing. We would like an unsupervised learning procedure to be applicable at successive layers, so that it can extract and explicitly represent progressively higher order features. If at one stage the algorithm could learn to explicitly represent continuous real-valued parameters such as relative depth, position and orientation of features in an image, subsequent learning could then discover higher order relations between these features, representing, for example, the location of object boundaries.

2 Spatial Coherence as an Objective Function

An unprocessed pixel image of a natural scene contains a wealth of information, but this information is in a complex form; a combination of intrinsic parameters such as depth, reflectance, and surface orientation, as well as effects of noise and illumination, all contribute to the intensity of each pixel. We would like an unsupervised processing stage to learn to encode this complex information in a simpler form, by representing the underlying parameters of interest explicitly. In order to constrain the kinds of regularities that our algorithm may discover, we propose that a good objective for perceptual learning is to extract higher-order features that exhibit simple coherence across space. In other words, we would like to represent the information in one spatially localized image region in a way that makes it easy to predict the representation of information in neighboring patches.

Some unpublished results of Peter Brown suggest that a good way to implement this general idea is to try to maximize the explicit mutual information between pairs of parameters extracted from adjacent but non-overlapping parts of the input. We derive a family of algorithms based on this principle, beginning with the simple case of binary parameters, and extending it to the case of continuously varying parameters.

The mutual information between two variables, a and b, is given by

$$I(a;b) = H(a) + H(b) - H(a,b)$$

where $H(a) = -<\log p(a)>$ is the entropy of a, and $H(a,b) = -<\log p(a,b)>$ is the entropy of the joint distribution of a and b. The equation shows that the mutual information between two variables can only be high if their joint distribution has low entropy, while each variable has high individual entropy. The latter requirement that each has high individual entropy is one advantage of mutual information over measures like the mean squared error between a and b: two variables can have minimal squared error simply by taking on constant, equal values, thus conveying no information. By maximizing the mutual information between variables extracted by two modules, each module is forced to convey a lot of information about some (spatially coherent) feature of its input. Figure 1 shows how this objective function can be used to train a multi-layer network.

2.1 Experiments with binary units

For the simple case of two binary probabilistic units, we can estimate the mutual information by discrete sampling of their activity values over a large set of input cases. Similarly, we can obtain an analytic expression for the derivative of their mutual information, as shown in Appendix A. The final expression we use for the partial derivative of the mutual information between the outputs of the ith and jth units with respect to the expected output of the ith unit on training case α, ${p_i}^{\alpha}$, is:

$$\frac{\partial I(y_i;y_j)}{\partial p_i^{\alpha}} \quad = \quad -P^{\alpha}\left[\log\frac{p_i}{p_{\bar{i}}} \; - \; {p_j}^{\alpha}\log\frac{p_{ij}}{p_{\bar{i}j}} \; - \; {p_{\bar{j}}}^{\alpha}\log\frac{p_{i\bar{j}}}{p_{\bar{i}\bar{j}}}\right] \tag{1}$$

where P^{α} is the probability of training case α, $p_i = <y_i>$ is the expected value of the ith unit's output averaged over the fluctuations for each training case and also over the whole ensemble of cases (when y_i is treated as a stochastic binary variable), $p_{\bar{i}} = <(1-y_i)>$, and $p_{ij} = <y_i y_j>$, etc.

We can see from equation 1 that in order for the ith unit to perform gradient ascent in $I(y_i;y_j)$, it needs to accumulate two ensemble-averaged statistics: the expected value that it is on, p_i, and the expected value of the joint event that it is on and the jth unit is also on, p_{ij}; from these two values the other terms in the individual and joint distributions can be derived (see Appendix A).

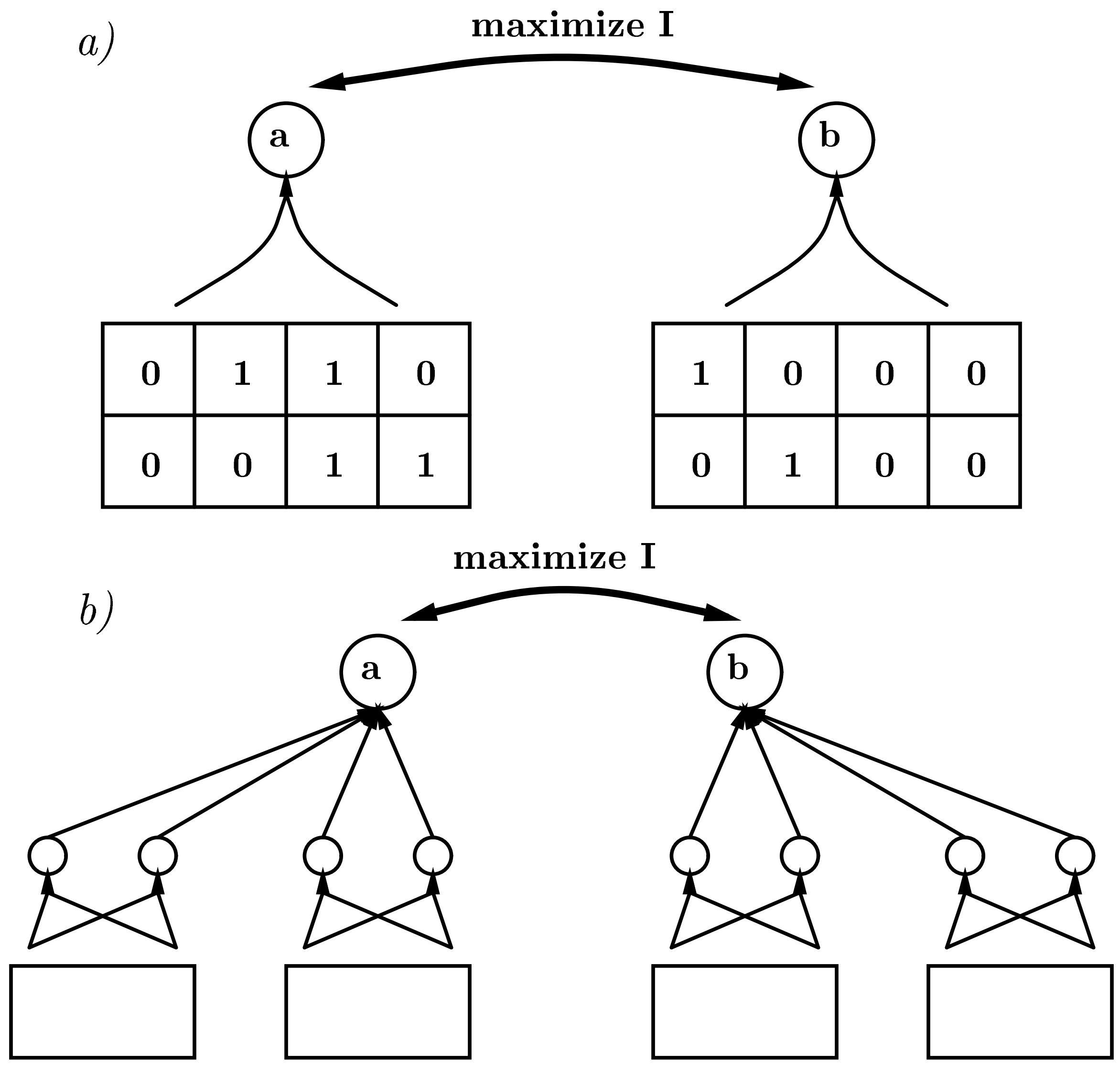


Figure 1: *a) Two units receive input from adjacent, non-overlapping parts of the image. The learning algorithm adjusts the weights of each unit to maximize the mutual information, over the ensemble of training cases, between the states of the two units. b) In a multi-layer version of this architecture, once the first layer of weights has been trained to extract some low order features, the next layer can hierarchically combine these noisy estimates into more accurate feature estimates.*

In our simulations we use a batch version of this algorithm: we make one sweep through the input cases to accumulate these probabilities, and a second sweep to accumulate the gradients. One could, however, implement an online version of the algorithm by accumulating time-averaged estimates of these probabilities, and using these approximate statistics to compute the weight update on each case.

We have found that this method works well for the task of discovering depth in an ensemble of very simple, binary random-dot stereograms, such as those shown in the input layer in Figure 1a. Each input vector consists of a one dimensional strip from the right image and the corresponding strip from the left image. The right image is purely random and the left image is generated from it by choosing, at random, a single global shift. So the input can be interpreted as an approximation to a one-dimensional stereogram of a fronto-parallel surface at an integer depth. The only local property that is invariant across space is the depth (i.e. the shift). Hence, if one unit looks at one area of the two images, and another unit looks at another area, the only way they can provide mutual information about each other's outputs is by representing the depth. We can measure the degree to which output units are tuned to depth by performing "simulated neuro-physiology": we present the network with a number of patterns at different depths, and observe how the activity levels of units vary with depth. A useful performance measure, indicative of the extent to which the outputs are shift-tuned, is the mutual information between each output unit's activity and the degree of shift.

We used two global shifts (one pixel rightwards or one pixel leftwards) operating on binary image strips. Each pair of strips was divided into n by 2 patches with a gap of one pixel between patches,[2] where the receptive field width n varied in different experiments. Each stochastic binary unit used the logistic non-linearity $\sigma(s) = 1/(1+e^{-s})$ to determine the probability of outputting

[2] With no gap between the receptive fields of two neighboring units, those units could simply learn the uninformative correlations between the upper (lower) rightmost bit of the left unit's field and the lower (upper) leftmost bit of the right unit's field, while ignoring the rest of their inputs.

a 1 as a function of its total weighted summed input s.[3] Units learned roughly by the method of gradient ascent, with the modification that the step size for each weight was truncated to lie within $[-0.1, 0.1]$. We found that in practice the gradients tended to increase by many orders of magnitude as the algorithm neared a solution, and this upper limit on the step size prevented the learning from "blowing up" in this situation.

We performed a number of preliminary experiments with the algorithm, varying the architecture and the training set size. With random patterns and a small training set there is a high probability that units will learn some of the random structure in the data in addition to the shift; as the number of training cases increases, sampling error decreases and units become more tuned to shift.

We can further increase the likelihood that shift will be learned, rather than random structure, by increasing the number of receptive fields (and hence the input size), because shift is the only feature common to the inputs of all units. When we extend the architecture shown in figure 1a to multiple receptive fields (with one unit receiving input from each receptive field) each unit now tries to maximize the sum of its pairwise mutual information with each of the other units. In this case, an interesting effect occurs as the learning proceeds: once some pair or subset of units within a layer has "caught on" to the shift feature, their mutual information gradients become very large, and convergence accelerates. This provides stronger shift-tuned signals to the other units, so that the effect rapidly spreads across the layer.

Since shift is a higher order feature of the input (i.e., it can be predicted from the products of pairs of pixel intensities but not by any linear combination of the individual intensities), a network with a single layer of weights cannot learn to become a perfect shift detector.[4] Hence, the problem requires a multilayer (nonlinear) network, so that with each successive layer, units can combine

[3]Rather than running stochastic simulations of probabilistic binary units, we actually used a mean field approximation in which the output of each unit on a particular case α is taken to be its expected value $\sigma(s^{\alpha})$. In practice this yields equivalent behaviour, but is much faster since we can obtain good estimates of the various probability statistics required for our learning algorithm in only one sweep through the training set.

[4]See Minsky and Papert(1969) for a thorough treatment of the issue of what can and cannot be learned by single layer networks.

the "votes" from several lower level feature detectors to make more robust predictions. Using a hierarchical architecture as shown in figure 1b, the partially shift-tuned noisy features extracted by the first layer from several patches of input can be combined by the second layer units, thereby increasing the signal to noise ratio in the prediction of shift. At the top layer, every "output" unit maximizes the sum of its mutual information with each of the others. In the middle layer, there are m clusters of units, each cluster receiving input from a different receptive field, and each output unit receiving input from a different set of clusters.

As mentioned above, the performance measure we use is the mutual information between the output units' activities and the shift, when shift is treated as a binary feature of the input pattern. For a network with two output units and four 2-unit clusters in the middle layer each with 2 by 5 receptive fields, the learning converged after about 300 passes through a training set of 500 patterns, and the output units learned to convey, on average (over 5 repetitions from different initial random weights), about .03 bits of information about the shift. As we increased the number of receptive fields, the ability of the network to learn shift increased dramatically. The most shift-tuned (and largest) network we experimented with on random shift patterns had 5 output units, each receiving input from two 2-unit clusters of units in the middle layer, and ten 2 by 4 receptive fields in the input layer. For this network, the learning took about 500 passes to converge on a training set of 1000 random patterns. The output units conveyed on average about .23 bits of information about the shift, and the most shift-tuned unit (over five runs from different initial random weights) conveyed .46 bits of information about the shift. Note that the maximum possible information that could be conveyed about a binary feature is 1 bit.

2.1.1 Using back-propagation to train the hidden layers

The multi-layer algorithm described above has the nice property that layers can be trained sequentially. Once the initial layer has partly learned the shift problem, subsequent layers can take advantage of this earlier learning and quickly become even more shift-tuned. However, there is no guarantee that the shift patterns will be perfectly classified by the top layer, since units in lower layers within a cluster may learn redundant features, which only apply to a subset of the input patterns. Since we only maximized the mutual information between *pairs* of hidden units, there was nothing to prevent many pairs from learning exactly the same feature. We can improve the performance of the network by applying our objective function between units at the top layer *only*, and using back-propagated gradients to train the middle layer. The back-propagated gradients send a more global training signal to the hidden units, which is best optimized by having hidden units differentiate to detect different features. In practice, as we shall see in the next section, a combination of the sequential, layer by layer, application of the same objective (as described in the previous section) and "top-down" or back-propagation training of the hidden units, seems to result in optimal performance.

Starting from random initial weights, the back-propagated gradients from the output units initially tend to be very small, in fact many orders of magnitude smaller than those near the solution. This causes convergence problems if the step size is a fixed proportion η of the gradient. What seems like a reasonable learning rate at the beginning will cause the learning to "blow up" near the solution. A more sophisticated optimization method would employ an adaptive learning rate, or better yet, a line search. For our early experiments using back-propagation, described in the remainder this section, we used what could be viewed as a very crude approximation to an adaptive learning rate, in which we empirically chose a very large value of η for the first 50 iterations, and then fixed it at a much smaller value for the remainder of learning. (Additionally, as before, the step size for each weight was truncated to lie within $[-0.1, 0.1]$.) We found that in practice this speeded up convergence considerably.[5]

We can remove the possibility that units may learn any of the random input structure by creating

[5]In our subsequent experiments on real-valued units (described in a later section in this paper), we employed a simple line search to accelerate convergence.

a complete set of unambiguous binary patterns. Using 2 by 4 receptive fields, there are $2^4 = 16$ possible left-shifted and 16 right-shifted patterns within a receptive field. We remove the 8 ambiguous patterns[6]. We used two modules, and created 16 bit patterns by presenting each possible combination of pairs of the 12 unambiguous 2 by 4 bit left-shifted patterns, and each combination of pairs of right-shifted patterns, yielding a total training set of 288 16-bit patterns.

The hierarchical architecture shown in figure 1b was helpful in dealing with noise when we trained layers sequentially, because the output units could compute a weighted average of information from multiple receptive fields. However, when we are back-propagating derivatives to hidden units, we can deal with this problem by simply adding more hidden units for each receptive field, because the back-propagated signal from an output unit provides pressure for the hidden units to extract different features. In our experiments on the unambiguous pattern set (and in the remaining experiments reported in this paper), instead of the hierarchical architecture used earlier, we used a modular architecture as shown in figure 2. Each 2 by 4 patch was used as input to a separate module that contained a layer of hidden units and one "output" unit.

Since we had removed the random aspect of the input patterns, we found we could now get by with a much smaller network having only two modules. When we trained our two module network on this complete set of patterns (with back-propagation to the hidden layers), we usually got (at least one of two) output units that were pure shift detectors within about 300 passes through the training set. The mutual information between the two output units occasionally approached the global maximum of 1 bit (corresponding also to 1 bit of information about the shift), although usually the network found only locally maximum solutions, as shown in figure 3a.

[6]The ambiguous patterns have 1111, 0000, 1010 or 0101 in the top half. Note that the left- and right-shifted versions of each of these (using shift with wrap-around within receptive fields) are indistinguishable. The use of wrap-around within each receptive field makes this example differ slightly from a simple version of a stereo task.

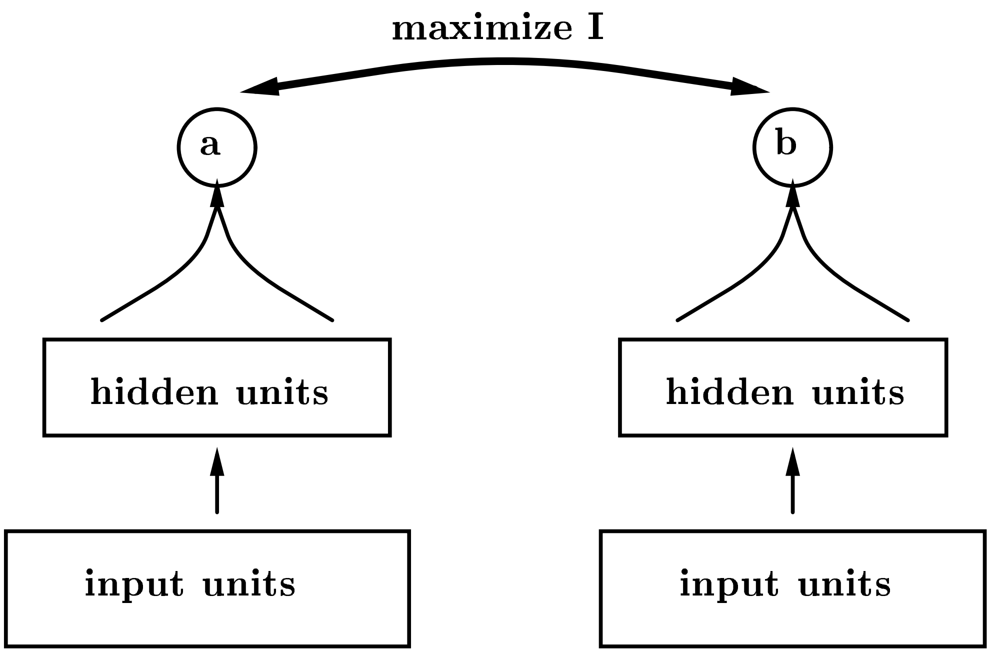


Figure 2: *Two modules that receive input from adjacent receptive fields. Each module has one layer of hidden units. The learning algorithm maximizes the mutual information between the states of the two output units, and the gradients are back-propagated to hidden units.*

2.1.2 Improving the performance with bootstrapping

The learning is rather slow for two reasons. First, we are not specifying desired values for the "output" units—we are only specifying that their pairwise mutual information should be high. The derivative of this objective function w.r.t a weight depends on *three* other layers of adaptive weights—one other layer in the same module and two layers in the adjacent module. So in this respect the difficulty of learning resembles back-propagation through four layers of weights. Second, with random starting weights, the initial gradient of the objective function is very small.

The performance of the algorithm, in terms of both convergence speed and quality of the final solution, is greatly improved by using a "bootstrapping" method. We start by applying our objective function between pairs of units within layers (as we did when in the hierarchical version) until these units are somewhat tuned to the shift. Then the gradients of the mutual information between the output units are much bigger and the objective function can be applied at that layer *only* and the derivatives back-propagated.[7]. More globally coherent information can now be provided to the hidden units that failed to find any useful features in the bootstrapping phase.

We ran simulations with three versions of the algorithm, one with bootstrapping, one with back-propagation, and one combined. We compared their performance on a network with two modules, using 8 hidden units and one output unit per module, on the complete, unambiguous pattern set described above. The three versions of the learning algorithm we used were:

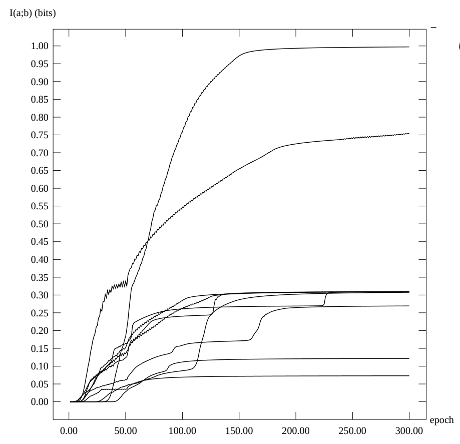
> **Version 1.** 50 bootstrapping learning iterations, in which each layer was trained to maximize information between pairs of units between modules, followed by 250 iterations with information maximization between the top two units and back-propagation of gradients to the hidden units.
>
> **Version 2.** 300 iterations in which each layer was trained to maximize information between pairs of units in different modules in the same layer, without back-propagation (as in the bootstrapping phase of Version 1).
>
> **Version 3.** 300 iterations with information maximization between the top two units and back-propagation to the hidden units (as in the back-propagation phase of Version 1).

Version 1 performed by far the best: in 48 out of 50 runs from different initial random weights, the network learned an exact solution to the shift problem, in which one or both output units predicted shift almost perfectly.[8] In Versions 2 and 3, while the top-level units nearly always became highly shift-tuned, only in 27 out of 50 and 30 out of 50 repetitions, respectively, did the network learn to perfectly separate the two classes of patterns. Figure 3 shows the learning curves for ten runs of Version 1 (bootstrapping) versus Version 3. We can see that in some cases the mutual information between the top two units asymptotically approaches the global optimum, while in other cases the

[7]One could alternatively apply both objectives simultaneously to the hidden units

[8]i.e. the mutual information between the unit's output and shift was greater than or equal to 0.6 nats = about .86 bits. In the remaining two runs, the output units were still both highly shift-tuned, but did not meet this criterion.

a)



b)

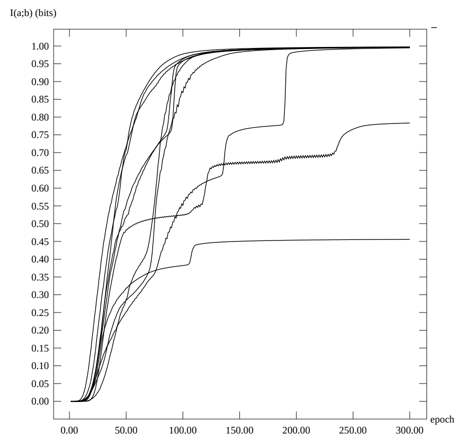


Figure 3: *Mutual information between two modules versus learning epochs on 288 binary shift patterns. a) shows learning curves using back-propagation alone, for 10 runs starting from different initial random weights, and b) shows learning curves using back-propagation preceded by 50 "bootstrapping" iterations, for 10 runs starting from the same initial weights as in a).*

procedure has become stuck in sub-optimal regions of weight space.

It appears that the bootstrapping procedure increases the likelihood that the network will converge to globally optimal solutions. The reason for this may be that because we are applying two different learning procedures to the hidden units, they have more opportunities to discover useful features at each stage. During the bootstrapping stage, a given pair of hidden units attempting to achieve high mutual information cannot learn to predict shift perfectly, since the shift problem is not learnable by a single layer of weights. However, some units do learn to be partially correlated with shift at the bootstrapping stage. This makes the job of a pair of output units in adjacent modules much easier, since they now have more spatially correlated inputs, which are at least partially shift-tuned. Now when we back-propagate the information gradients to the hidden units, those hidden units which failed to learn anything useful in the bootstrapping stage are under increased pressure to become shift-tuned.

Not only does the bootstrapping procedure increase the overall quality of the solution, but on more difficult learning problems it substantially reduces the learning time. We find that for the continuous version of our algorithm (described in the next section), on stereograms with continuously varying depths, when the hidden units are initially trained with bootstrapping, the subsequent learning with back-propagation of gradients is accelerated by roughly an order of magnitude.

3 Modules with real-valued outputs

The binary probabilistic output units we used in our initial experiments tend to learn solutions with large weights, resulting in sharp decision boundaries. The drive toward maximal entropy causes a unit to adopt its two possible states with equal probability over the ensemble of training cases, but the drive toward minimal joint entropy causes it *on each particular case* to be either strongly on or strongly off, so that it can agree with its neighbor with high probability. This "within-case" tendency towards strongly binary behaviour of the output units has some undesirable consequences. As the magnitudes of the weights increase, the unit's sigmoidal probabilistic response function sharpens into a step function. On each particular case, the unit's response is at one of the extremes of this function, where its slope is virtually zero. This means that effectively the unit can neither adapt any further, nor can its output be used to indicate its degree of uncertainty about its decision, since it has virtually become a linear threshold unit.

This binary behaviour makes it difficult to represent depth in more realistic images that contain smoothly curved surfaces which have real-valued depths. In this case, we would like units to learn to encode continuous ranges of depths with real values. Ideally, we would like a group of units to form a population code for depth, in which each unit has a roughly Gaussian-response to some limited range of disparities. Such a mechanism appears to be employed in biological systems (Lehky and Sejnowski, 1990), and could be achieved, for example, using the continuous interval encoding scheme proposed by Saund (1986). We are investigating the use of an alternative scheme in our current research; for now, however, let us assume that it is possible for a single unit's real-valued output to reliably encode depth, given that the input patterns have only a small range of disparities.

In the following simulations, we use random dot stereograms of curved surfaces, as shown in figure 4, and modules with deterministic, real-valued outputs that learn to represent real-valued depths (disparities) with sub-pixel accuracy. A slice of a curved surface is generated by randomly choosing a collection of control points (scaled to lie within a fixed depth range), and fitting a curve through these points using spline interpolation, as shown in figure 4. The curve can be thought of as the depth of a gently curved surface as a function of the (horizontal) x-coordinate, for a fixed y-coordinate. Points are pseudo-randomly scattered on the curve by dividing the x-axis into a fixed number of intervals, and randomly placing a point within each interval, to create a pattern of random features.

Stereo images are formed from this pattern by taking two parallel projections from slightly different viewing angles.[9] Finally, we add Gaussian blur (using a Gaussian with standard deviation roughly equal to one image pixel) to the stereo images and integrate these blurred patterns within equally spaced intervals to create the discretized real-valued n-dimensional input vectors. The Gaussian blurring causes a single point from the random dot pattern to spread over several pixels in the final image. The final pair of n-pixel stereo patterns is treated as a 2 by n image, which is divided into smaller stereo receptive fields. The spacing of the random features is chosen so that, on average, each receptive field will contain about two features, and never less than 1 feature. (We discuss the problem of missing or unreliable data in the final section.)

The analytic expression for the mutual information between two real-valued variables a and b is $I(a;b) = -[\int \log p(a) + \int \log p(b) - \int\int \log p(a,b)]$. To extend our algorithm to real-valued units, we must make some simplifying assumptions about the probability distributions of the variables, in order to obtain a tractable approximation to this expression. We start by making the following very simple coherence assumption (which will be relaxed later): There is some locally detectable

[9]The parameters of the projection (focal length, eye offset and image width) were chosen so that the maximum disparity was 6 minutes of visual arc in the "imaging system", corresponding to a final disparity of 2 image pixels.

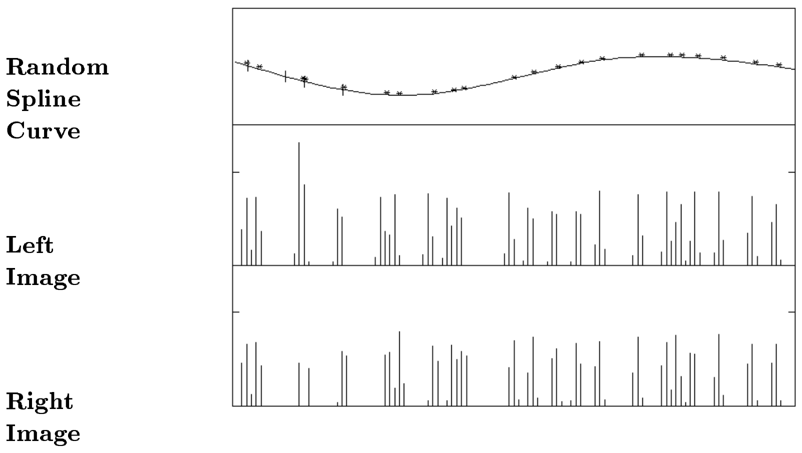


Figure 4: *Part of a cubic spline fitted through seven randomly chosen control points, with randomly located features scattered on it, and the "intensity" values in the two images of the surface strip. The images are made by taking two slightly different parallel projections of the feature points, filtering the projections through a Gaussian, and sampling the filtered projections at equally spaced sample points. The sample values in corresponding patches of the two images are used as the inputs to a module. The boundaries of two neighboring patches are shown on the spline.*

parameter which is approximately constant for nearby patches. So, given two modules A and B that receive input from neighboring patches, we want their outputs a and b to be approximately equal. We can think of b as a signal that we are trying to predict and a as a version of that signal that is corrupted by additive, independent, Gaussian noise. If we assume that both a and b have Gaussian distributions, the information that a provides about b is determined by the log ratio of two variances:

$$I_{a;b} = 0.5 \log \frac{V(signal + noise)}{V(noise)}$$

The reader who is familiar with information theory will recognize that the above is a standard expression for the information transmission rate of a noisy channel when the signal and noise have Gaussian distributions (Shannon, 1949).

In our model, where a is a noisy estimator of b, the information rate is:

$$I_{a;b} = \log \frac{V(a)}{V(a-b)}$$

So, for a to provide a lot of information about b we need a to have high variance but $a - b$ to have low variance. For symmetry, we actually optimize the following objective function (where κ is a small constant which prevents the information measure from becoming infinitely large):

$$I^*_{a;b} = I_{a;b} + I_{b;a} = \log \frac{V(a)}{V(a-b)+\kappa} + \log \frac{V(b)}{V(a-b)+\kappa}$$

In the above model, modules A and B are actually making inconsistent assumptions about each other's output distributions (Ralph Linsker, *personal communication*). Theoretically, $I_{a;b}$ should be equal to $I_{b;a}$. However, each module is assuming that it is conveying a noisy version of some signal, and that the other is conveying the pure signal. Under these assumptions, A would predict its own output to have higher variance than b, while B would predict its own output to have higher variance than a. Thus, the assumptions cannot both be valid except in the limit as the variance of $a-b$ approaches zero. However, in practice this is not a problem, because each module is minimizing $V(a-b)$, so our model becomes more accurate as learning proceeds.

To avoid the asymmetry in our mutual information measure, another possibility would be to assume that both a and b are noisy versions of the same underlying signal, each with independent additive Gaussian noise. This alternative model has the advantage that modules A and B are now making consistent assumptions about each other, but the disadvantage that we cannot exactly compute the information that either of the modules alone transmits about the underlying signal. As an approximation, we could optimize the expression

$$\begin{aligned} \log \frac{V(a+b)}{V(a-b)} &= \log \frac{V(signal + noise_a + signal + noise_b)}{V(noise_a - noise_b)} \\ &= \log \frac{V(2 * signal + noise_a + noise_b)}{V(noise_a + noise_b)} \end{aligned}$$

as suggested by Allan Jepson (*personal communication*). This measure tells how much information the sum $a+b$ conveys about the signal. In our simulations, we use the former model, with the above expression for I^* and we find that it gives good results in practice.

The derivation for the gradient of our objective function I^* is given in Appendix B. The final expression we use in our learning algorithm for the partial derivative of $I^*(y_i; y_j)$, the mutual information between the outputs of the ith and jth units with respect to the output of the ith unit on case α, ${y_i}^\alpha$, is:

$$\frac{\partial I^*(y_i; y_j)}{\partial {y_i}^\alpha} \quad = \quad \frac{2}{N}\left[\frac{{y_i}^\alpha - < y_i >}{V(y_i)} - 2\frac{({y_i}^\alpha - {y_j}^\alpha) - < y_i - y_j >}{V(y_i - y_j) + \mu}\right] \qquad (2)$$

where $< y_i >$ is the output of the ith unit averaged over the ensemble of N training cases.

The objective we have proposed for the continuous case is closely related to a statistical method called ACE(Hastie and Tibshirani, 1990) (for Alternating Conditional Expectation), a nonlinear generalization of regression. ACE minimizes the squared error between a single nonlinearly transformed predictor variable y and an additive nonlinear transformation of a set of m observed variables x_i, over an ensemble of n observations:

$$E = \sum_{i=1}^{n}(f(y_i) - \sum_{j=1}^{m}\phi(x_{ij}))^2$$

subject to the constraint that $f(y)$ has unit variance. While ACE's underlying model differs from ours (we compute nonlinear transformations of *two sets* of variables - the two input patches, and we do not use an additive model, since the input components are not in general independent), both objectives are equivalent to maximizing the correlation between two nonlinearly transformed variables.

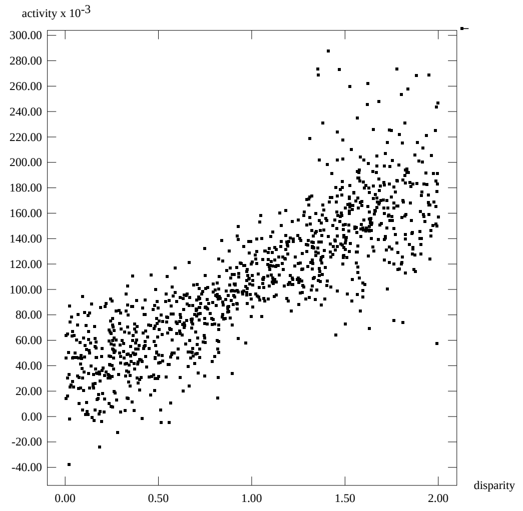
3.1 Discovering real-valued depth for fronto-parallel surfaces

Figure 4 shows how we generate stereo images of curved surface strips. The same technique can be applied to generate images of fronto-parallel planar surfaces. Using 1000 training cases of this simpler type of input, we trained a network that contained 10 modules each of which tried to maximize I^* with the immediately adjacent modules. Each module had a single linear output unit, and 10 nonlinear hidden units.

Each update of the weights involves two complete passes through the training set. In the first pass, we compute the mean and variance of each output value and the mean and variance of each pairwise difference between output values given the current weights (see in Appendix B). In the second pass we compute the derivatives of I^* (w.r.t. each unit's output) as shown in equation 2 for the output units of each pair of adjacent modules. We then back-propagate these terms to the hidden layer, and for each unit, we use these derivatives to accumulate dI^*/dw for all weights. Then we update all the weights in parallel. After each weight update, we average corresponding weights in all the modules in order to enforce the constraint that every module computes exactly the same function of its input vector. (This idea of constraining the weights has been used effectively by le Cun et al. (1989) in a supervised learning task).

Without these equality constraints among groups of weights, the algorithm does not converge; each module tries to learn some of the random structure that it finds in common with the adjacent neighbor, and this effect "swamps out" the effect of disparity. This is because it is much easier for two units to learn correlations between pairs of their input lines than it is to learn higher order structure that would eventually lead to more globally optimal solutions. When we average weight updates among groups of units receiving input from different input patches, the effect of this low order noise is reduced, since only the disparity is common to all input patches. (Note that we dealt with this problem differently in the binary case, by allowing each unit to maximize mutual information with *many* other units rather than just its immediate neighbors.)

a)



b)

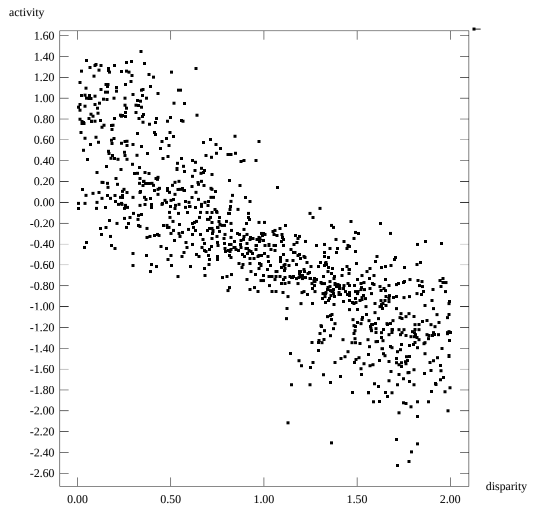


Figure 5: *The activity of the linear output unit of a module (vertical axis) as a function of the disparity (horizontal axis) for all 1000 test cases using planar surface strips. a) shows the response for a network trained with 10 adaptive nonlinear hidden units per module, and b) for a network trained with 100 nonadaptive Gaussian units per module.*

To accelerate the convergence, we used a simple line search. Each learning iteration consists of two phases: *1)* computation of the gradient, as described above, and *2)* computation of an appropriate learning rate η. At the beginning of each iteration, η is increased by a factor of 1.05. In phase 2, we first increment the weights by a proportion η of the gradient, and then if the average amount of mutual information between adjacent output units has decreased, we repeatedly backtrack, halving η each time, until the average information is greater than or equal to its previous value.

Before applying this learning algorithm, we first trained the hidden layer only, with 100 bootstrapping learning iterations, maximizing I^* between pairs of hidden units in adjacent modules.[10] We then trained the 10 modules, as described above, for 1400 iterations. Each module became highly tuned to disparity, as shown in figure 5a.

[10]As in the binary case, we do not want two hidden units within the same module to directly or indirectly optimize their mutual information (for they could learn trivial solutions). They could do so indirectly by each optimizing their mutual information with the same unit in another module. Hence we arbitrarily number the n units within the hidden layer of each module, and only allow hidden units in different modules to communicate with each other if they have been given the same number.

3.2 Speeding the learning using radial basis functions

We experimented with an additional method to accelerate learning, in which the first hidden layer of adaptive units in each module is replaced by a large number of non-adaptive radial basis functions (RBFs) (Moody and Darken, 1989). Each radial basis unit has a "center" $\overline{\mathbf{x}}$, that is equal to a typical input vector selected at random, and gives an output y, which is a Gaussian function of the distance between the current input vector and the unit's "center":

$$y = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{||\mathbf{x}-\overline{\mathbf{x}}||}{\sigma^2}}$$

All the Gaussians have the same variance σ^2, which is chosen by hand.[11] Each module had 8 by 2 input units connected to a layer of 100 non-adaptive RBFs. Every module used exactly the same set of RBFs so that we could constrain all the modules to compute the same function.

We compared the performance of the learning with a non-adaptive hidden layer of RBFs, versus the algorithm described in the previous subsection with an adaptive layer of nonlinear hidden units, on the fronto-parallel surfaces. We found that with the RBFs, the learning took only about 50 iterations to converge, a speedup of roughly an order of magnitude. However, there is some loss of precision in the solution; units still became highly depth-tuned, but had greater variance, as shown in figure 5b.

In the remaining experiments described in this paper for discovering depth in stereo pairs of curved surface strips, we used an initial layer of RBFs, choosing to sacrifice some precision for increased learning speed.

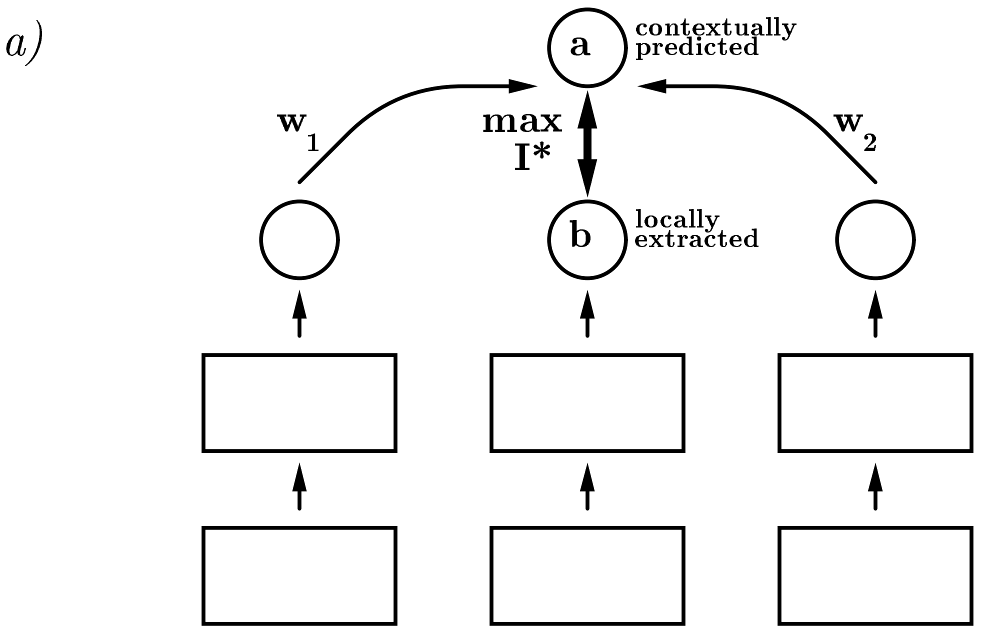
[11]The variance was 0.1. The input vectors had peak values of about 0.3.

3.3 More complex types of coherence

So far, we have described a very simple model of coherence in which an underlying parameter at one location is assumed to be approximately equal to the parameter at a neighboring location. This model is fine for fronto-parallel surfaces but it is far from the best model of slanted or curved surfaces. Fortunately, we can use a far more general model of coherence in which the parameter at one location is assumed to be an unknown linear function of the parameters at nearby locations. The particular linear function that is appropriate can be learned by the network.

We used a network of the type shown in figure 6a (but with 10 modules and with a contextual predictor unit for all modules except the two at the ends). We tried to maximize I^* between the output of each module and the contextual prediction of this output that was produced by computing a linear function of the outputs of one adjacent module on each side. We used weight averaging to constrain this interpolating function to be identical for all modules. We also back-propagated the error derivatives through the interpolating weights. Before applying this new objective function, we first used a bootstrapping stage in which we maximized I^* between adjacent pairs of modules as before, for 30 learning iterations.

After having been trained for 100 iterations on 3000 patterns, the two weights learned for the interpolating function were $.55, .54$. The output of these units is similar to the response profile of units trained on fronto-parallel surfaces, shown in figure 5b, but even more finely depth-tuned. Thus, the interpolating units have learned that the depth at one patch on a planar surface can be approximated by the average of the depths of surrounding patches.



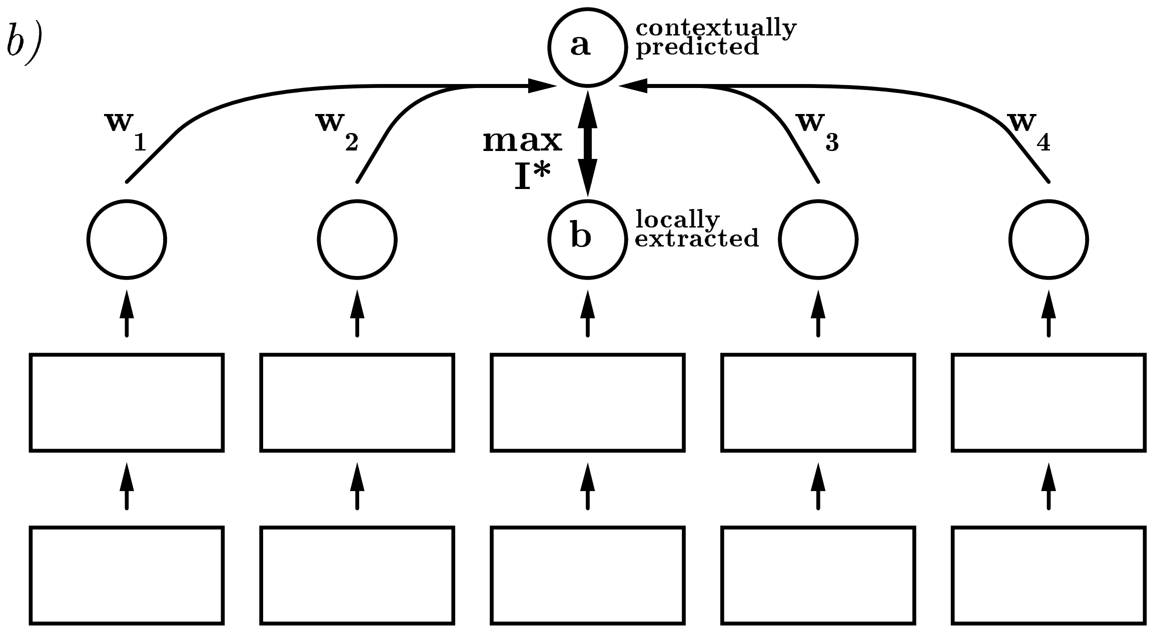


Figure 6: *Networks in which the goal of the learning is to maximize the information between the output of a local module and the contextually predicted output that is computed by using a linear function of the outputs of nearby modules. a) The network used for planar surface interpolation. b) The network used for curved surface interpolation.*

3.4 Discovering a surface interpolator for curved surfaces

As we introduce curvature in the surfaces, the prediction of depth from neighboring patches becomes more difficult; at regions of high curvature, a simple average of the depths of 2 adjacent patches will under- or over-estimate the true depth. In this case, a better interpolator would base its predictions on more than two local measurements of depth, thereby taking curvature into account.

We trained a network of 10 modules on 1000 of the stereograms of curved surface strips, using the same architecture and objective function as for the planar surface task, for 30 iterations. We then added an interpolating layer; this time, however, the contextual prediction of a given module was a linear function of the outputs of *two* adjacent modules on either side, as shown in figure 6b. After 100 iterations, the four weights learned for the interpolating function were $-.04, .64, .65, -.04$. Positive weights are given to inputs coming from the immediately adjacent modules, and smaller negative weights are given to inputs coming from the other two neighbors. The activity of these units is well tuned to disparity, as shown in figure 7. Given noise-free depth values, the optimal linear interpolator for the surface strips we used is approximately $-.2, .7, .7, -.2$. But with noisy depth estimates it is better to use an interpolator more like the one the network learned because the noise amplification is determined by the sum of the squares of the weights.

4 Discussion

The goal of our depth interpolating unit was to discover how to predict one value from a linear combination of nearby values, by maximizing the log ratio of its output variance to the variance of the prediction error. Another way to pose this problem is to find an invariant relation between the predicted value and the other values by learning a linear combination of *all* the values, such that the combined value always equals zero (Richard Durbin, *personal communication*). In this case,

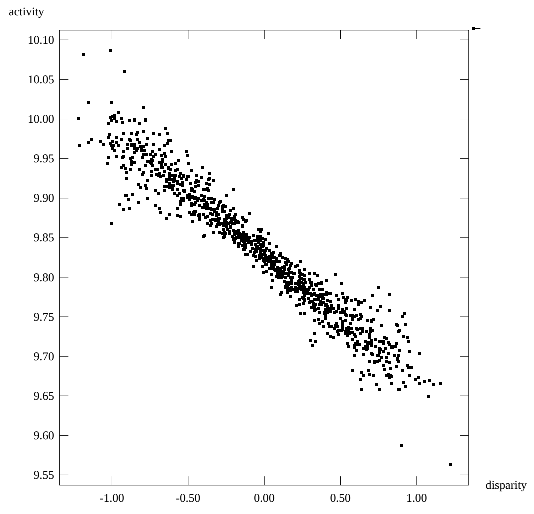


Figure 7: *The output of a unit (vertical axis) as a function of the local disparity (horizontal axis) for a test set of 1000 patterns, when trained on 1000 curved surface strips. The unit learned to predict the depth locally extracted from one module as a linear function of the outputs of the two adjacent modules on either side.*

we would minimize the variance of the predicting unit's output about zero, while maximizing the variances of the individual inputs. This amounts to discovering invariant higher-order properties by learning invariance detectors that have low variance even though their input lines have high variances and large weights. One attractive aspect of this view is that the actual output of an invariance detector would represent the extent to which the current input violates the network's model of the regularities in the world. This is an efficient way of transmitting information about the current input. Furthermore, it has been suggested that the computation of invariant features about the world plays a fundamental role in human pattern recognition (Dodwell, 1983).

An invariance detector that minimizes the ratio of its output variance divided by the variance that would be expected if the input lines were independent Gaussian variables is a real-valued, deterministic, approximate version of the G-Maximization learning procedure (Pearlmutter and Hinton, 1986) which finds regularities by maximizing the extent to which the independence assumption is incorrect in predicting the output of a unit. It also has an interesting relation to Linsker's learning procedure (Linsker, 1988). Linsker assumes the variances and covariances of the activities on the input lines to a unit are fixed (because he does not backpropagate derivatives) and he shows that, with the appropriate Gaussian assumptions, the information conveyed by the unit about its input vector is maximized by using weights which *maximize* the ratio of the output variance divided by the sum of the squares of the weights.

4.1 Learning multidimensional features

We have described the learning procedure for modules which each have a single real-valued output. For modules with several real-valued outputs, the natural way to generalize the objective function is to use the information rate for multi-dimensional Gaussians, where the variance is replaced by the determinant of the covariance matrix (Shannon, 1949). Zemel and Hinton (1991) have applied a version of this algorithm to explicitly extract information about the parameters of the viewing transform in images of simple two-dimensional objects.

4.2 Modeling discontinuities

We have also ignored the ubiquitous problem of discontinuities. Images of real scenes have strong local coherence punctuated by discontinuities. We do not want our learning procedure to smear out the strong local coherence by trying to convey information across the discontinuities. We would prefer a module to make accurate predictions in continuity cases and no predictions in other cases rather than making rather inaccurate predictions in all cases.

We can achieve this by allowing a module A to use a slightly more complicated model to predict b, the depth of a nearby patch (or a linear combination of several nearby depths). Instead of assuming that the predicted signal b is always drawn from a Gaussian distributed identically to its own output a (plus noise), module A models b's distribution as a mixture of two Gaussians, as shown in figure 8.

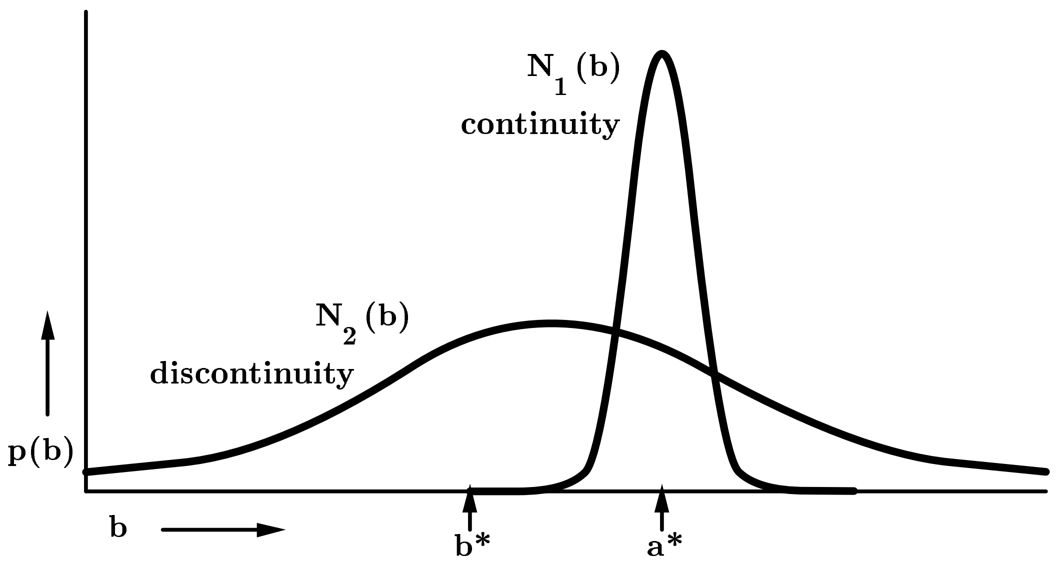


Figure 8: *The probability distribution of b, p(b), is modeled as a mixture of two Gaussians:* $N_1(b)$ *with mean* $= a^*$ *and small variance, and* $N_2(b)$ *with large variance. Sample points for a and b,* a^* *and* b^* *are shown. In this case,* a^* *and* b^* *are far apart so* b^* *is more likely to have been drawn from* N_2.

For each input pattern, A assumes that b was either drawn from a normal distribution $N_1(b, \mu_1, {\sigma_1}^2)$ with mean $\mu_1 = a$ and small variance ${\sigma_1}^2$ (a continuity case) or from another less predictable distribution $N_2(b, \mu_2, {\sigma_2}^2)$ with much greater variance (a discontinuity case).

The posterior probability that the current case α is a continuity example can be computed by comparing the probability densities, under both the continuity and discontinuity distributions, of the observed output of module B, as is done in the EM algorithm[12](Dempster, Laird and Rubin, 1977):

$$p_1(b^\alpha) = \frac{\pi_1 N_1(b, \mu_1, {\sigma_1}^2)}{\sum_{i=1}^{2} \pi_i N_i(b, \mu_i, {\sigma_i}^2)} \tag{3}$$

[12]EM is an iterative optimization procedure which can be used to find a (locally) maximum likelihood model using a finite mixture of arbitrary component densities for a set of incomplete data points.

where π_i's are the mixing proportions. The proportions can be fixed, or can be estimated from the data as in the EM algorithm, e.g. $\pi_1 = \sum_{\alpha=1}^{n} {p_1}^{\alpha}/n$. We can now optimize the *average* information a transmits about b, by optimizing the product of the mixing proportion π_1 of the continuity Gaussian and the information conveyed given that we have a continuity case:

$$I^{**} = -\pi_1[<\log p(a)> + <\log p_1(b)> - <\log p_1(a,b)>]$$

The contribution to the accumulated gradient is thus made proportional to the probability that the current case is a continuity example. This means that clear cases of discontinuity do not affect the weights learned by the continuity model.

One way to implement this idea is with an iterative two-phase learning procedure, in which we first adjust the weights while freezing the mixture model parameters, and then freeze the weights while optimizing the mixture model. After each sweep through the training data, we could use a gradient method to update the weights in the network, and then use several iterations of EM to reestimate the variances and mixing proportions of the continuity and discontinuity Gaussians. This only approximately follows the gradient of the above objective function, since the implicit assumption is made that the weights and model parameters are independent. In practice, there are difficulties in getting this method to converge, probably due to the invalidity of the independence assumption.

A better approach would be to have a means of estimating the mixture model parameters which is independent of the weights in the stereo network. For example, an additional "expert network" could estimate the likelihood that each input pattern represents a continuity or discontinuity case, as in the competing experts model of Jacobs et al.(1991).

4.3 Population codes for depth

An alternative way to model discontinuities falls out naturally if we use a population code (as discussed at the beginning of section 2) to represent depth, rather than using the activity of each output unit to encode the entire range of disparities. If each member of a group of units is tuned to a slightly different disparity, with a roughly Gaussian response, and each receives input from roughly the same region, we can interpret the response of each unit in this population (when normalized by the total population response) as the likelihood that it detected its preferred disparity. This population code can also be viewed as a mixture of Gaussians model for depth within a local image region. When one unit is most strongly active, its preferred disparity is, with high probability, close to the true disparity. If there are two sharp peaks in the population's depth response, this is strong evidence of a depth discontinuity within the region (assuming that we do not have two transparent surfaces).

Another problem we have not dealt with is that of missing or noisy data. When we are interpreting the activity of a unit directly as a real-valued code for depth, there is no obvious way to represent the certainty of the local estimate. The population code handles this problem in a natural way. When the local data is missing or very ambiguous, the population will have uniform low activity over all units. Thus, the entropy of the activity distribution represents uncertainty.

A simple way to ensure that the population encodes a probability distribution is to make the expected output of each unit be:

$$p_i = \frac{e^{s_i/T}}{\sum_j e^{s_j/T}}$$

where s_i is the total input to the ith unit in the population. This also allows an additional way of explicitly modeling uncertainty by using a critic to compute an appropriate value of T. For example, when the relative phase of two bandpass filtered images is used to extract disparity, there are some easily detectable situations in which the disparity estimate is unreliable (Allan Jepson, *personal* *communication*, 1989). A critic could detect these situations and set T to a high value. In principle, the critic's weights could also be learned by back-propagation. It remains to be seen whether this works in practice.

4.4 Discovering other kinds of coherence

In this chapter, we have described a learning procedure for discovering coherence across space in visual images, and demonstrated its effectiveness in learning about depth from stereo images. There are many other spatially coherent features which could be learned, such as illumination, texture, orientation and curvature. There is also coherence within and between signals acquired in other sensory modalities, such as the correspondence between tactile and visual information.

Finally, in virtually every sensory modality, there is coherence across time. In the visual domain, for example, in sequences of images at successive points in time, there is temporal coherence in features such as the size and shape of moving objects. To train a network to learn temporal coherence, we could apply essentially the same learning procedure as before, by maximizing the mutual information between a network's outputs at successive time frames.

5 Conclusions

We have described the broad goals of unsupervised learning as twofold: firstly, we would like it to capture as much as possible of the interesting structure in the sensory input; secondly, we wish to represent that structure in a form which facilitates subsequent supervised learning. In particular, we would like the unsupervised learning to discover a transformation of the sensory input into an explicit representation of the underlying *causes* of the sensory input. The visual system, for example, transforms the patterns of light impinging on the retina into important features of the world such as object boundaries, motion and depth.

With relatively few constraints, simple statistical methods such as clustering and principal components analysis can be applied to images to reduce noise, and represent simple low order features. As we further constrain the learning, we can discover more specific, higher order features and form explicit representations of important parameters of the world.

One way to constrain the learning is to find features that are spatially predictive, allowing a network to discover spatially coherent properties of visual images, such as depth, texture and curvature. The procedure we have described has several appealing properties. First, it builds into the objective function (and the architecture) a type of prior knowledge that is strongly constraining but widely applicable to perceptual tasks. Second, using the bootstrapping approach it may be possible to train deep networks fairly rapidly, provided the domain is such that higher order features that exhibit long-range coherence can be built out of lower order features that exhibit shorter range coherence. We have illustrated this idea by first training a network to discover depth in small patches of stereo images, and then training depth-interpolating units which combine depth estimates from several local patches.

Acknowledgements

We thank Peter Brown, Francis Crick, Richard Durbin, Allan Jepson, Barak Pearlmutter and Steve Nowlan for helpful discussions. This research was supported by grants from DuPont, the Ontario Information Technology Research Center, and the Canadian Natural Science and Engineering Research Council. Geoffrey Hinton is a fellow of the Canadian Institute for Advanced Research.

Appendix A

The Mutual Information Gradient in the Binary Case

We assume a simple model neuron which computes a nonlinear probabilistic function σ of its real-valued total input s_i. For a particular input case α, the output of the ith unit is a binary variable $y_i \in \{0, 1\}$, which is on with probability ${p_i}^\alpha = \sigma({s_i}^\alpha)$ and off with probability $1 - {p_i}^\alpha$. The total input to a unit is computed as a weighted sum of the activities on its input lines: ${s_i}^\alpha = \sum_k w_{ik} {y_k}^\alpha$.

We can compute the probability that the ith unit is on or off by averaging over the input sample distribution, with P^α being the prior probability of an input case α:

$$\begin{aligned} p_i &= \sum_\alpha P^\alpha {p_i}^\alpha \\ p_{\bar{i}} &= 1 - p_i \end{aligned}$$

Similarly, for a pair of binary units in a feed-forward network which are not connected to one another (but may share the same inputs), and hence compute independent functions on a particular case, we can compute the four possible values in the joint (discrete) probability distribution of their outputs, y_i and y_j, as follows:

$$\begin{aligned} p_{ij} &= \sum_\alpha P^\alpha {p_i}^\alpha {p_j}^\alpha \\ p_{\bar{i}j} &= p_j - p_{ij} \\ p_{i\bar{j}} &= p_i - p_{ij} \\ p_{\bar{i}\bar{j}} &= 1 - p_j - p_i + p_{ij} \end{aligned}$$

The partial derivatives of the (expected) individual and joint probabilities with respect to the expected output of the ith unit on case α are:

$$\begin{aligned}
\frac{\partial p_i}{\partial p_i{}^\alpha} &= P^\alpha \\
\frac{\partial p_i}{\partial p_{\bar{i}}{}^\alpha} &= -P^\alpha \\
\frac{\partial p_{ij}}{\partial p_i{}^\alpha} &= P^\alpha p_j{}^\alpha \\
\frac{\partial p_{\bar{i}j}}{\partial p_i{}^\alpha} &= -P^\alpha p_j{}^\alpha \\
\frac{\partial p_{i\bar{j}}}{\partial p_i{}^\alpha} &= P^\alpha (1 - p_j{}^\alpha) \\
\frac{\partial p_{\bar{i}\bar{j}}}{\partial p_i{}^\alpha} &= -P^\alpha (1 - p_j{}^\alpha)
\end{aligned}$$

The amount of information transmitted by the ith unit on case α when it is on is:

$$I(y_i{}^\alpha = 1) = -\log p_i{}^\alpha$$

If the log is base 2, the units of information are bits; for the natural log, the units are nats. From here on, we use *log* to denote the natural log. When the value of $y_i{}^\alpha$ is unknown, the average information (or equivalently the entropy or uncertainty) in $y_i{}^\alpha$ is:

$$H(y_i{}^\alpha) = -[p_i{}^\alpha \log p_i{}^\alpha + p_{\bar{i}}{}^\alpha \log p_{\bar{i}}{}^\alpha]$$

Averaged over all input cases, the entropy of y_i is:

$$H(y_i) = -\langle \log p(y_i) \rangle = -[p_i \log p_i + p_{\bar{i}} \log p_{\bar{i}}]$$

The mutual information between the outputs of the ith and jth units, y_i and y_j, is:

$$I(y_i; y_j) = H(y_i) + H(y_j) - H(y_i, y_j)$$

where $H(y_i, y_j)$ is entropy of the joint distribution $p(y_i, y_j)$:

$$\begin{aligned} H(y_i, y_j) &= -\langle \log p(y_i, y_j) \rangle \\ &= -[p_{ij} \log p_{ij} + p_{i\bar{j}} \log p_{i\bar{j}} + p_{\bar{i}j} \log p_{\bar{i}j} + p_{\overline{ij}} \log p_{\overline{ij}}] \end{aligned}$$

The partial derivative of $I(y_i; y_j)$ with respect to the expected output of unit i on case α, ${p_i}^{\alpha}$, can now be computed; since $H(y_j)$ does not depend on ${p_i}^{\alpha}$, we need only differentiate $H(y_i)$ and $H(y_i, y_j)$.

$$\begin{aligned} \frac{\partial H(y_i)}{\partial {p_i}^{\alpha}} &= \frac{\partial}{\partial {p_i}^{\alpha}}[-(p_i \log p_i + p_{\bar{i}} \log p_{\bar{i}})] \\ &= -[\frac{\partial p_i}{\partial {p_i}^{\alpha}} \log p_i + \frac{p_i}{p_i}\frac{\partial p_i}{\partial {p_i}^{\alpha}} + \frac{\partial p_{\bar{i}}}{\partial {p_i}^{\alpha}} \log p_{\bar{i}} + \frac{p_{\bar{i}}}{p_{\bar{i}}}\frac{\partial p_{\bar{i}}}{\partial {p_i}^{\alpha}}] \\ &= -[\frac{\partial p_i}{\partial {p_i}^{\alpha}}(\log p_i + 1) + \frac{\partial p_{\bar{i}}}{\partial {p_i}^{\alpha}}(\log p_{\bar{i}} + 1)] \\ &= -[P^{\alpha}(\log p_i + 1) - P^{\alpha}(\log p_{\bar{i}} + 1)] \\ &= -P^{\alpha} \log \frac{p_i}{p_{\bar{i}}} \end{aligned}$$

$$\begin{aligned} \frac{\partial H(y_i, y_j)}{\partial {p_i}^{\alpha}} &= \frac{\partial}{\partial {p_i}^{\alpha}}[-(p_{ij} \log p_{ij} + p_{i\bar{j}} \log p_{i\bar{j}} + p_{\bar{i}j} \log p_{\bar{i}j} + p_{\overline{ij}} \log p_{\overline{ij}})] \\ &= -\left[\frac{\partial p_{ij}}{\partial {p_i}^{\alpha}} \log p_{ij} + \frac{p_{ij}}{p_{ij}}\frac{\partial p_{ij}}{\partial {p_i}^{\alpha}} + \frac{\partial p_{\bar{i}j}}{\partial {p_i}^{\alpha}} \log p_{\bar{i}j} + \frac{p_{\bar{i}j}}{p_{\bar{i}j}}\frac{\partial p_{\bar{i}j}}{\partial {p_i}^{\alpha}} \right. \\ &\quad \left. + \frac{\partial p_{i\bar{j}}}{\partial {p_i}^{\alpha}} \log p_{i\bar{j}} + \frac{p_{i\bar{j}}}{p_{i\bar{j}}}\frac{\partial p_{i\bar{j}}}{\partial {p_i}^{\alpha}} + \frac{\partial p_{\overline{ij}}}{\partial {p_i}^{\alpha}} \log p_{\overline{ij}} + \frac{p_{\overline{ij}}}{p_{\overline{ij}}}\frac{\partial p_{\overline{ij}}}{\partial {p_i}^{\alpha}}\right] \end{aligned}$$

$$
\begin{aligned}
&= -\left[\frac{\partial p_{ij}}{\partial {p_i}^{\alpha}}(\log p_{ij}+1)+\frac{\partial p_{\bar{i}j}}{\partial {p_i}^{\alpha}}(\log p_{\bar{i}j}+1)\right. \\
&\qquad \left.+\frac{\partial p_{i\bar{j}}}{\partial {p_i}^{\alpha}}(\log p_{i\bar{j}}+1)+\frac{\partial p_{\overline{ij}}}{\partial {p_i}^{\alpha}}(\log p_{\overline{ij}}+1)\right] \\
&= -\Big[P^{\alpha}{p_j}^{\alpha}(\log p_{ij}+1)-P^{\alpha}{p_j}^{\alpha}(\log p_{\bar{i}j}+1) \\
&\qquad +P^{\alpha}(1-{p_j}^{\alpha})(\log p_{i\bar{j}}+1)-P^{\alpha}(1-{p_j}^{\alpha})(\log p_{\overline{ij}}+1)\Big] \\
&= -P^{\alpha}[{p_j}^{\alpha}\log\frac{p_{ij}}{p_{\bar{i}j}}+{p_{\bar{j}}}^{\alpha}\log\frac{p_{i\bar{j}}}{p_{\overline{ij}}}]
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial I(y_i;y_j)}{\partial {p_i}^{\alpha}} &= \frac{\partial H(y_i)}{\partial {p_i}^{\alpha}}-\frac{\partial H(y_i,y_j)}{\partial {p_i}^{\alpha}} \\
&= -P^{\alpha}\left[\log\frac{p_i}{p_{\bar{i}}}-{p_j}^{\alpha}\log\frac{p_{ij}}{p_{\bar{i}j}}-{p_{\bar{j}}}^{\alpha}\log\frac{p_{i\bar{j}}}{p_{\overline{ij}}}\right]
\end{aligned}
$$

Appendix B

The Mutual Information Gradient in the Continuous Case

In the real-valued case, we assume each unit deterministically computes its output y_i according to some (linear or nonlinear) function $y_i = \sigma(s_i)$ of its total input s_i. The total input to a unit on case α is computed as a weighted sum of the activities on its input lines: ${s_i}^{\alpha} = \sum_k w_{ik} {y_k}^{\alpha}$. The goal of the ith unit is to transmit as much information as possible about some underlying signal to be extracted from its input. In the simplest model, unit i assumes that its own output y_i is equal to the (Gaussian) signal plus some Gaussian noise, and that the jth output y_j is exactly that signal. Under this model, the mutual information between y_i and y_j is:

$$I_{y_i;y_j} = \log \frac{V(signal + noise)}{V(noise)} = \log \frac{V(y_i)}{V(y_i - y_j)}$$

For symmetry, we optimize the following approximation to the information rate:[13]

$$I^*_{y_i;y_j} = I_{y_i;y_j} + I_{y_j;y_i} = \log \frac{V(y_i)}{V(y_i - y_j)} + \log \frac{V(y_j)}{V(y_i - y_j)}$$

For unit i to maximize the above information measure, it must store four statistics: its output mean $\langle y_i \rangle$ and variance $V(y_i)$, and the mean and variance of the difference of the units' outputs, $\langle y_i - y_j \rangle$ and $V(y_i - y_j)$. These statistics are computed in an initial pass through the training set:

$$\langle y_i \rangle = \frac{1}{N} \sum_{\alpha=1}^{N} {y_i}^{\alpha}$$

$$V(y_i) = \frac{1}{N} \sum_{\alpha=1}^{N} ({y_i}^{\alpha})^2 - \langle y_i \rangle^2$$

[13]Actually, we add a small constant κ to the denominator terms $V(y_i - y_j)$; this prevents the information measure from growing infinitely large and stabilizes the convergence. Thus a minor change to the above derivation is required when we add in this stabilizing term.

$$\langle y_i - y_j \rangle = \frac{1}{N} \sum_{\alpha=1}^{N} ({y_i}^{\alpha} - {y_j}^{\alpha})$$

$$V(y_i - y_j) = \frac{1}{N} \sum_{\alpha=1}^{N} ({y_i}^{\alpha} - {y_j}^{\alpha})^2 - \langle y_i - y_j \rangle^2$$

Now we can compute the partial derivative of the information the ith unit conveys about the jth unit with respect to the output of the ith unit on case α:

$$\begin{aligned}
\frac{\partial {I^*}_{y_i;y_j}}{\partial y_i^{\alpha}} &= \frac{\partial}{\partial y_i^{\alpha}} \left[\log \frac{V(y_i)}{V(y_i - y_j)} \; + \; \log \frac{V(y_j)}{V(y_i - y_j)} \right] \\
&= \frac{\partial \log V(y_i)}{\partial y_i^{\alpha}} - 2 \frac{\partial \log V(y_i - y_j)}{\partial y_i^{\alpha}} \\
&= \frac{1}{V(y_i)} \frac{\partial V(y_i)}{\partial y_i^{\alpha}} \; - \; \frac{2}{V(y_i - y_j)} \frac{\partial V(y_i - y_j)}{\partial y_i^{\alpha}} \\
&= \frac{1}{V(y_i)} (\frac{2}{N} {y_i}^{\alpha} \; - \; \frac{2}{N} \langle y_i \rangle) \; - \; 2 \left[\frac{1}{V(y_i - y_j)} (\frac{2}{N} ({y_i}^{\alpha} - {y_j}^{\alpha}) \; - \; \frac{2}{N} \langle y_i - y_j \rangle) \right] \\
&= \frac{2}{N} \left[\frac{{y_i}^{\alpha} - \langle y_i \rangle}{V(y_i)} - 2 \frac{({y_i}^{\alpha} - {y_j}^{\alpha}) - \langle y_i - y_j \rangle}{V(y_i - y_j)} \right]
\end{aligned}$$

References

Baldi, P. and Hornik, K. (1989). Neural networks and principal components analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58.

Blakemore, C. and Cooper, G. (1970). Development of the brain depends on the visual environment. *Nature*, 228:477–478.

Blakemore, C. and van Sluyters, R. (1975). Innate and environmental factors in the development of the kitten's visual cortex. *Journal of Physiology*, 248:663–716.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294.

Carpenter, G. and Grossberg, S. (1987). ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23). Special issue on neural networks.

Crawford, M., Smith, E., Harwerth, R., and von Noorden, G. (1984). Stereoblind monkeys have few binocular neurons. *Investigative Opthalmology and Visual Science*, 25(7):779–781.

Crawford, M., von Noorden, G., Meharg, L., Rhodes, J., Harwerth, R., Smith, E., and Miller, D. (1983). Binocular neurons and binocular function in monkeys and children. *Investigative Opthalmology and Visual Science*, 24(4):491–495.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Proceedings of the Royal Statistical Society*, B-39:1–38.

Derrington, A. (1984). Development of spatial frequency selectivity in striate cortex of vision-deprived cats. *Experimental Brain Research*, 55:431–437.

Dodwell, P. (1983). The lie transform group model of visual perception. *Perception and Psychophysics*, 34(1):1–16.

Földiák, P. (1990). Forming sparse representations by local anti-hebbian learning. *Biological Cybernetics*, 64:165–170.

Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136.

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

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models.* Chapman and Hall, London.

Hirsch, H. V. B. and Spinelli, D. (1970). Visual experience modifies distribution of horizontally and vertically oriented receptive fields in cats. *Science*, 168:869–871.

Hubel, D. and Wiesel, T. (1963). Receptive fields of cells in striate cortex of very young, visually inexperienced kittens. *Journal of Neurophysiology*, 26:994–1002.

Huber, P. (1985). Projection pursuit. *The Annals of Statistics*, 13(2):435–475.

Jacobs, R. A. (1987). Increased rates of convergence through learning rate adaptation. Department of Computer and Information Sciences COINS-TR-87-117, University of Massachusetts, Amherst, Ma.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1).

Kohonen, T. (1982). Clustering, taxonomy, and topological maps of patterns. In Lang, M., editor, *Proceedings of the Sixth International Conference on Pattern Recognition*, Silver Spring, MD. IEEE Computer Society Press.

Lang, K. J. and Hinton, G. E. (1988). A time-delay neural network architure for speech recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Back-propagation applied to handwritten zipcode recognition. *Neural Computation*, 1(4):541–551.

Lehky, S. R. and Sejnowski, T. J. (1990). Neural model of stereoacuity and depth interpolation based on a distributed representation of stereo disparity. *The Journal of Neuroscience*, 10:2281–2299.

Linsker, R. (1986a). From basic network principles to neural architecture: Emergence of orientation-selective cells. *Proceedings of the National Academy of Sciences USA*, 83:8390–8394.

Linsker, R. (1986b). From basic network principles to neural architecture: Emergence of orientation columns. *Proceedings of the National Academy of Sciences USA*, 83:8779–8783.

Linsker, R. (1986c). From basic network principles to neural architecture: Emergence of spatial opponent cells. *Proceedings of the National Academy of Sciences USA*, 83:7508–7512.

Linsker, R. (1988). Self-organization in a perceptual network. *IEEE Computer,* March, 21:105–117.

Linsker, R. (1989). Designing a sensory processing system: What can be learned from principal components analysis? Technical Report RC14983 (66896), IBM.

Merzenich, M. (1987). Dynamic neocortical processes and the origins of higher brain functions. In Changeux, J. P. and Konishi, M., editors, *The Neural and Molecular Bases of Learning, Dahlem Konferenzen*, pages 337–358. John Wiley and Sons Limited.

Merzenich, M. M., Allard, T., Jenkins, W. M., and Recanzone, G. (1988). Self-organizing processes in adult neo-cortex. In W. von Seelen, U. M. L. and Shaw, G., editors, *Organization of neural networks: Structures and models.* VCH publishers.

Miller, K., Keller, J., and Stryker, M. (1989). Ocular dominance column development: Analysis and simulation. *Science*, 245:605–615.

Miller, K. D. (1990). Correlation-based models of neural development. In Gluck, M. A. and Rumelhart, D. E., editors, *Neuroscience and Connectionist Theory.* Lawrence Erlbaum Associates.

Minsky, M. L. and Papert, S. (1969). *Perceptrons.* MIT Press, Cambridge, Mass.

Moody, J. and Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, *1*(2):281–294.

Nowlan, S. J. (1990). Maximum likelihood competitive learning. In Touretzky, D. S., editor, *Neural Information Processing Systems, Vol. 2*, pages 574–582, San Mateo, CA. Morgan Kaufmann.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273.

Oja, E. (1989). Neural networks, principal components, and subspaces. *International Journal Of Neural Systems*, 1(1):61–68.

Olson, C. and Freeman, R. (1980). Profile of the sensitive period for monocular deprivation in kittens. *Experimental Brain Research*, 39:17–21.

Pearlmutter, B. A. and Hinton, G. E. (1986). G-maximization: An unsupervised learning procedure. In *Snowbird Conference on Neural Networks and Computing.*

Plaut, D. C., Nowlan, S. J., and Hinton, G. E. (1986). Experiments on learning by back-propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh PA 15213.

Poggio, T. (1989). A theory of networks for approximation and learning. A.I. Memo No. 1140, C.B.I.P. Paper No. 31, MIT Artifiacal Intelligence Laboratory, and Center for Biological Information Processing, Whitaker College.

Renals, S. and Rohwer, R. (1989). Phoneme classification experiments using radial basis functions. In *the IEEE International Conference on Neural Networks.*

Rumelhart, D. E. and Zipser, D. (1986). Feature discovery by competitive learning. In D. E. Rumelhart, J. L. M. and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I. Bradford Books, Cambridge, MA.

Sanger, T. (1989a). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473.

Sanger, T. (1989b). An optimality principle for unsupervised learning. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 1, pages 11–19, Denver 1988. Morgan Kaufmann, San Mateo.

Sanger, T. D. (1989c). Optimal unsupervised learning in feedforward neural networks. M.Sc. Thesis, Department of Electrical Engineering and Computer Science, MIT.

Saund, E. (1986). Abstraction and representation of continuous variables in connectionist networks. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 638–644, Los Altos, CA. Morgan Kaufmann.

Shannon, C. E. (1949). A mathematical theory of communication. part iii. *Bell System Technical Journal*, XXVIII:623–656.

Stryker, M. P. and Harris, W. A. (1986). Binocular impulse blockade prevents the formation of ocular dominance columns in cat visual cortex. *The Journal of Neuroscience*, 6(8):2117–2133.

Sur, M., Garraghty, P., and Roe, A. (1988). Experimentally induced visual projections into auditory thalamus and cortex. *Science*, 242:1437–1441.

Von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in striate cortex. *Kybernetik*, 14:85–100.

Watrous, R. L. and Shastri, L. (1987). Learning phonetic features using connectionist networks: An experiment in speech recognition. In *IEEE International Conference on Neural Networks.*

Wiesel, T. N. and Hubel, D. H. (1965). Comparison of the effects of unilateral and bilateral eye closure on cortical unit responses in kittens. *Journal of Neurophysiology*, 28:1029–1040.

Zemel, R. S. and Hinton, G. E. (1991). Discovering viewpoint-invariant relationships that characterize objects. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances In Neural Information Processing Systems 3*, pages 299–305. Morgan Kaufmann Publishers.