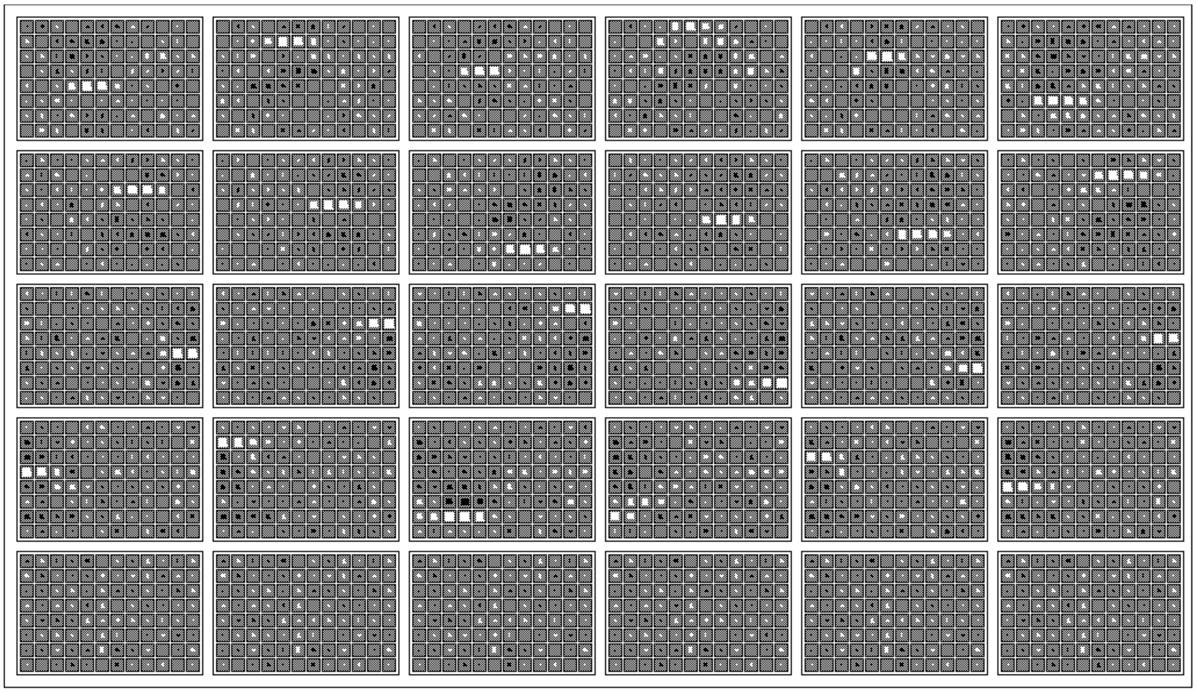


Figure 5.10: This figure shows the outgoing weights for a CVQ network with 5 VQs and 6 units per VQ. The first four rows depict the weights of the units in the first four VQs. Each VQ responds to curve segments between a particular pair of control points. The weights of the fifth VQ, shown in the final row, are approximately equal for each unit. The output of the VQ is consistent on each case, so the activity cost is minimized, and since the outgoing weights are small, the variance is also minimized. This VQ is thus effectively eliminated.

($\overline{y_j}$) decreases. Thus when a unit is active on a given case ($y_j \approx 1.0$), the activity cost (Equation 5.3) is high. Empirically, we have found that a 4-VQ network with 12 units per VQ can reach a generalization length of 22 bits per input, but additional units tend to either get shut down or make the algorithm more likely to end up in a local minimum.

A CVQ network with 5 VQs also develops a good representation of the spline dataset. With additional VQs, the network can again improve the quality of its reconstruction, but the cost of encoding the activities increases more dramatically. Since the VQs are treated independently with respect to the activity cost, each VQ is costly. A network with 5 VQs typically forms one of two representations, or ends up in a local minimum containing some hybrid of the two. The first representation is the same segment-based representation found in the 4-VQ case, where the extra VQ is eliminated. An example is shown in Figure 5.10. The final generalization length of this solution approximately equals that of the 4-VQ case, since the extra VQ is basically free. The second representation actually uncovers the true causes of the data: each VQ comes to represent the height of one of the 5 control points along the spline image. The network now forms a reconstruction by concatenating the heights of each control point. The units within a VQ do not typically only capture the height of a control point, but also the position of the neighboring control point(s) on either side, much as the units in the 4-VQ case captured some regularities of segments on either side.

While this solution captures the underlying generator of the images, the generalization length is not much different than that of the 4-VQ case. For example, a 5-VQ CVQ network with 12 units per VQ that forms a control-point representation has a generalization length of 19 bits. We can likely attribute the similarity in the generalization lengths between the segment and control-point representations to the process by which the images were generated. Even though each image had five degrees of freedom, these 5 degrees were then translated into a spline composed of four curve-segments connecting the 5 points. Thus in terms of the image itself, the 4-segment representation was as efficient and informative as the 5-control-point representation. In fact, with fewer units per VQ in a network that forms the control-point representation, the generalization length approaches 30 bits.

A second explanation for why the control-point representation is not clearly superior to the segment representation is due to the structure of the network. In this network, the reconstruction is a simple linear transformation of the representation units' activities. The linear transformation is an adequate method of combining the segments in the 4-VQ representation. Transforming from a representation of the heights of the control points to the spline, however, necessitates re-generating the cubic B-spline procedure. A network with a layer of units between the representation units and the outputs could learn to more closely approximate this procedure. We return to this point below.

Finally, a network with more than 5 VQs and at least 6 units per VQ typically forms a representation analogous to either the segment or control-point representations, and then eliminates the excess VQs. Here the cost of the extra VQs outweighs any gain in reconstruction ability due to the additional representation units.

#### Comparison to standard methods

The representations discovered by the CVQ network are generally more efficient and informative than those found by standard methods. For example, a standard soft competitive learning network (which can also be viewed as a CVQ network with a single VQ) with 24 units lacks the cooperative nature of a CVQ representation. Since there are not enough units to devote one to each curve, the network typically activates several units to represent each input. This leads both to poor reconstruction and high variance in the output. The final description length of this network is 40 bits per image, 30 of which are devoted to the summed-squared-difference between the actual and desired outputs.

A PCA-type reconstruction network (a linear autoencoder with no competition in the representation layer) also cannot match the CVQ representations. For such a network with 24 representation units, for example, the reconstruction error is slightly lower than that of the 4 VQ by 6 units CVQ network (0.11 per image versus 0.14), but the cost of encoding the values of those 24 representation units is much greater than the cost of encoding the activities in the CVQ network. In the PCA network, the representation units assume continuous values. These values can be encoded to some fixed precision, but the induced error in this quantization would have to be added to the generalization length (see Section 2.3.1). This demonstrates a problem with learning algorithms for data compression that do not take into account the cost of encoding the activities in the objective. The final generalization length of this network would clearly exceed the 25 bits per image of the comparable CVQ network.

Finally, CVQ also outperforms the standard method of dividing the image into 4 pieces and then using one VQ for each piece, even when we generously divide the image into pieces in the most helpful possible way (which depends on the data). We test this by training a CVQ network with fixed connectivity: one VQ connects to the 8x3 block of pixels between the first two control points, the second VQ to the next block, etc. This network learns features that resemble the horizontal segments learned in the fully-connected 4-VQ network (see Figure 5.8). Here, however, the network cannot take advantage of an important aspect of the input: the heights of two neighboring segments are correlated, and are in fact joined since the curve is continuous. The final description length of the network is 5 bits more than that of the fully-connected CVQ network, all of which is in the reconstruction cost. The fully-connected network learns about and takes advantage of the smooth joining of the segments, as the weights extend beyond the 8x3 block of pixels between two points, and thus forms a better representation.

## 5.5 Discussion

The CVQ algorithm is capable of forming non-redundant, informative representations of the datasets presented in this chapter. This algorithm extends the representational scheme introduced in the previous chapter, which assumed that the representation would consist of a few active features from a wide vocabulary of non-redundant features of the data. In CVQ, these features are expanded into a set of mutually exclusive alternative values, and a representation is assembled from a set of these 1-of-$N$ choices. This algorithm is thus complementary to the approach taken in the mixture of experts algorithm (Jacobs et al., 1991) where the units within an "expert" module cooperate to attempt to account for the data, and then the different experts compete to take responsibility. In CVQ, the units within each module compete to account for the data, and then the modules cooperate to reconstruct the input.

This algorithm also can be seen as a method of extending standard competitive learning (CL) to form distributed representations. Previous methods aimed at this goal have encountered difficulties. Rumelhart and Zipser (1985) suggested using several independent competitive pools, each of which may form its own partition of the input space. This often fails because all pools will discover the same partition if it is clearly better than others. In CVQ, we place competitive learning in a reconstruction paradigm, which forces the pools to collectively encode the input. Meanwhile, the coding cost for the activities squeezes redundancy out of the representation, encouraging the algorithm to eliminate any aspect of the representation not contributing to the reconstruction. These two pressures thus combine to force different pools to encode different components of the input, taking responsibility for separate subspaces of the full data space.

Mozer (1991) proposed an iterative version of competitive learning (ICL) which iteratively applies a single CL, or VQ step to the residual from the previous step. This also forces the different pools to capture different aspects of the data. Yet the iterative nature of this algorithm means that these aspects will be ordered in significance or saliency. This algorithm would not be able to extract structure at the same level of description, such as the segments of the splines or the two components of the dual-component inputs.

We are currently investigating two extensions to CVQ. The first extension would allow the algorithm to learn to represent different types of objects in separate sets of VQs. One set of VQs could represent the different elements of a spline as above, while a second set could represent the elements of some other shape. Such a network could be trained on all shapes at once, or incrementally. For example, the CVQ networks described above with more than 4 VQs eliminate the excess VQs. If the network is then shown examples of a new shape, these VQs could be recruited to represent it. The current method we are considering for this extension would allow all the VQs within a set to in effect share a none-of-the-above unit. If this unit is active, then no other bits are required to specify the activities of units in the set. Each input is assumed to contain an instance of one of the shapes accounted for by the various sets. The none-of-the-above units for these sets would compete, so that only one could be inactive on any given input case. The network could then efficiently represent several different objects in a non-redundant manner.

This would not necessarily be the most efficient representation for the various objects. If a VQ within a set captures some significant feature or dimension of the object, such as its color or position, a better representation might allow these features to be shared among different sets of VQs all describing objects to which that feature is relevant. A hierarchical representation allows for this sort of feature sharing. Extending CVQ to a hierarchical scheme is a natural elaboration of the algorithm. A higher-level VQ could capture redundancies between structures in a lower level. Alternatively, these additional layers need not be VQs, but may compute some nonlinear function of the representation units' activities, such as was discussed with respect to the control-point representation in the previous section.

The same difficulty, however, applies to extending a CVQ network to more than three layers as applied to the networks described in the previous chapter: Since the additional layers would be nonlinear, we cannot compute the expected reconstruction cost and activity cost exactly. While these can be computed when the layer(s) above the representation are linear, as in the networks described above, approximations or Monte Carlo simulations must be used if these layers are nonlinear. Preliminary results, where the output layer of a CVQ network is composed of sigmoid units, indicate that the algorithm can still operate effectively under such an approximation.

Chapter 6

# Developing Population Codes

In the CVQ algorithm, each vector quantizer comes to represent an independent feature of the underlying process that generates a set of data. Each unit within a particular vector quantizer represents a possible value of that feature.

In many instances, rather than discrete possibilities, these values may better be described as values along a continuum, and the feature as an underlying *dimension* of the dataset. We saw an example of this in the splines dataset described in the previous chapter, where the units within a VQ came to represent various heights of a control point. Each unit described a discrete alternative, while the true underlying variable was continuous: the real-valued, random position of the point along the $y$-axis.

This notion of continuous underlying dimensions of a dataset forms the basis for the representational scheme of the algorithm described in this chapter, the third algorithm derived from the MDL framework. The following section presents some of the motivations and novel aspects of this algorithm. We then derive the activity-cost and the reconstruction-cost, and present the results of applying the algorithm to three different datasets.

## 6.1 Introduction

### 6.1.1 Constraint surfaces

In the previous chapters, we have shown that if we assume that some common origin or relation exists between the data, then discovering the characteristics or constraints under which the source that generates the data set operates can lead to succinct descriptions. In many cases, these constraints are continuous-valued dimensions, so that each datum is a point on an underlying *constraint surface*. For example, a set of points constrained to lie on a simple curve can be described by their $(x, y)$-coordinates. A more compact description of each point is its position $s$ along the curve. This simple example shows that uncovering the underlying constraint can lead to *dimensionality-reduction*, or a recoding of the data in a lower-dimensional space.

Another example is shown in Figure 6.1. If we imagine a high-dimensional input representation produced by digitizing these images, many bits would be required to describe each instance of the hippopotamus. But after seeing several such instances, we can describe any particular one by describing a canonical hippo, and then just specifying the four 2D viewing parameters: $(x, y)$-position, orientation, and scale. A third example exists in the biological domain, where recent studies have found that in monkey motor cortex, the direction of movement in 3D space of a monkey's arm is encoded by the activities of populations of cells, each of which responds based on its preferred direction of motion (Georgopoulos, Schwartz and Kettner, 1986). Georgopoulos showed that averaging each cell's preferred motion directions weighted by its activity accurately predicts the direction of movement that the animal makes. In each of these examples, knowledge of an underlying constraint surface, generally of lower dimensionality than the input space, allows for compact descriptions of stimuli or responses.

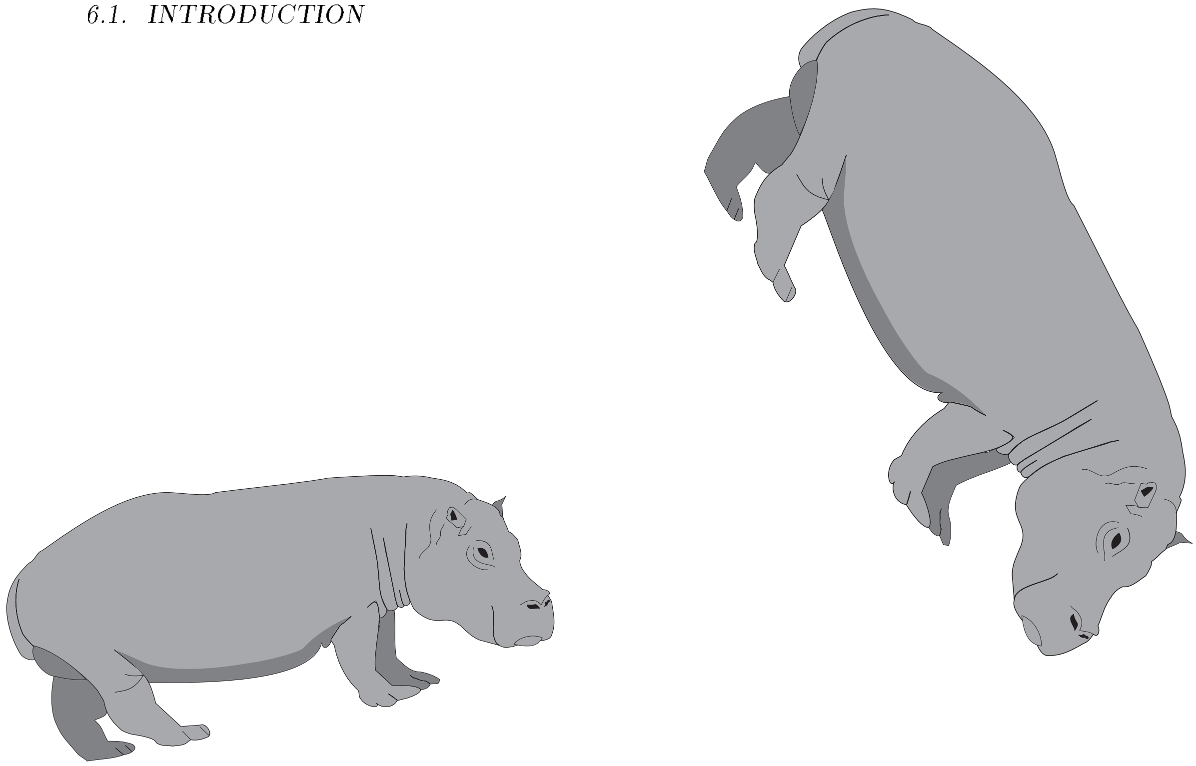


Figure 6.1: Two example images where only a few dimensions of variability underlie seemingly unrelated vectors of pixel intensity values.

### 6.1.2 Representing underlying dimensions using population codes

The goal is to find and represent the $m$-dimensional constraint surface underlying the $n$-dimensional data set, given a set of samples drawn from the constraint surface.

Representations of a point on this surface by the values of a set of units divide into two primary categories. The first type involves continuous-valued units, where the real-valued activity of a given unit describes the position of a particular instance on the surface. The second form of representation is to divide the surface into patches, and represent each patch with a separate unit.[1] Rather than this strictly local representation, where each point on the surface is represented by a single unit, this second form can be made more distributed by having the patches overlap, so that each point is represented by several units.

In this algorithm, we employ this type of more distributed form of representation, a coarse-coding of the underlying space, or a *population code*: each unit represents a local range of points on a surface, and the activity of several units together convey a particular point.

Population codes have several merits. First, they are another form of sparse distributed coding, and thus enjoy the same advantages that have been mentioned in previous chapters. The resulting representation is fault-tolerant, in that each unit need not function perfectly in order to represent a set of inputs. More than one value can be represented at a time, provided that they are sufficiently

[1] These two different forms of representing continuous values are also known as *variable* and *value* units (Ballard, 1986), where a variable unit's activity is proportional to a scalar parameter, while a value unit responds to a particular value or range of values of the variable.

far apart on the constraint surface, and each unit has a small dynamic range. Also, as first suggested by Barlow (1961), this form of representation seems to be a property of most cortical cells.

In addition to these advantages, a population code as described above reflects structure in the input, forming a topographic map of the underlying surface where nearby points on the surface are assigned similar representations. The representation therefore captures the surface in a smooth and stable manner. And since the pattern of activity of several units convey a point on the surface, the representation is not limited in the way that a more localist representation, such as CVQ, is to describing one of a set of discrete alternatives, but instead can describe values along a continuous dimension.

### 6.1.3 Introducing redundancy into the representation

Population codes, however, would seemingly be difficult to develop in the MDL framework. In the algorithm described in Chapter 4, the activity of each representation unit was communicated separately and independently, while in Chapter 5, each pool of units was communicated independently, and the units within a pool competed to be active. Both of these algorithms, and any encoding scheme where the activities are encoded independently, tend to lead to factorial codes; since mutual information between representation units will cause redundancy in the communicated message, the pressure to keep the messages short will squeeze out the redundancy.

In a population code, however, the activities of the representation units would be highly correlated. The representation divides the surface into overlapping patches, so units representing nearby patches will have similar activities. The aim of this third algorithm is to show how the MDL approach can be used to develop representations of constraint surfaces underlying datasets in which the activities of the representation units are highly correlated.

## 6.2 The Model

### 6.2.1 The representation scheme

A key idea in this algorithm is that population codes involve *three* quite different spaces:

1. the input-vector space;

2. the hidden, or representation-vector space; and

3. a low-dimensional space which we call the *implicit space*.

For example, in a Kohonen network (Kohonen, 1982), the representation unit with the greatest total input has an activity of one, while the others are zero (see Section 3.3.3). The representation-vector is thus a delta function. Yet these representation units are also assigned positions in implicit space based on a neighborhood function that determines the degree of interaction between a pair of units according to their distance. The active unit then maps the input to a particular point in this implicit space. Similarly, in a 1-dimensional version of the elastic net algorithm (Durbin and Willshaw, 1987), the representation units are assigned positions along a ring; these units are then pulled towards the inputs, but are also pulled towards their ring neighbors. In the Travelling

Salesman Problem, for example, the inputs are cities, and representation units adjacent in implicit space represent consecutive cities in the tour, so that the active unit for a given input city describes its order in the tour.

Representation units therefore play a dual role in population codes, as their pattern of activity not only conveys a representation of the input, but also a position in this implicit space. In a learning algorithm for population codes, this implicit space is intended to come to smoothly represent the underlying dimensions of variability in the inputs, i.e., the constraint surface.

Consider an example where we define a one-dimensional implicit space to represent points lying along a curve. The inputs here are the $(x, y)$-coordinates of each point, so the input-vector space is two-dimensional. The hidden-vector space is $n$-dimensional, where $n$ is the number of representation units. Each representation unit is then trained to respond to inputs in a range of positions along the curve, such that the response of neighboring units in implicit space are correlated. This creates a population code in the one-dimensional implicit space, which comes to correspond to the underlying constraint surface, in this case the position $s$ along the curve.

In order to represent this implicit space, we must incorporate more structure into the representation scheme. In most topographic-map formation algorithms, such as the Kohonen and elastic net algorithms, the topology of this implicit space is defined by neighborhood relations. In our algorithm, we use a more explicit representation. Each hidden, or representation unit has weights coming from the input units that determine its activity level. But in addition to these weights, it has another set of adjustable parameters that represent its coordinates in the implicit space. These coordinates, along with the weights, are learned based on a single objective function. To determine what implicit location is represented by a vector of representation activities, we average together the implicit coordinates of the units, weighting each coordinate vector by the activity level of the unit.

Suppose, for example, that each representation unit is connected to an 8x8 retina and has 2 implicit coordinates that represent the position of a particular kind of shape on the retina. Given a population of such units all broadly-tuned to different positions we can represent any particular instance of the shape by the relative activity levels in the population. If we plot the representation activity levels in the implicit space (not the input space), we would like to see a bump of activity of a standard shape (e.g., a Gaussian) whose center represents the instantiation parameters of the shape (Figure 6.2 depicts this for a 1D implicit space).

If the activities form a perfect Gaussian bump of fixed variance in implicit space we can communicate them by simply communicating the coordinates of the mean of the Gaussian; this is very economical if there are many less implicit coordinates than representation units.

It is important to realize that the activity of a representation unit is actually caused by the input-to-hidden weights, but by setting these weights appropriately we can make the activity match the height under the Gaussian in implicit space. If the activity bump is not quite perfect, we must also encode the *bump-error*—the misfit between the actual activity levels and the levels predicted by the Gaussian bump. The cost of encoding this misfit is what forces the activity bump in implicit space to approximate a Gaussian. We describe this bump-error in more detail below.

Finally, as in the earlier algorithms, the representation units act as the input to the output units of the network. The reconstruction-error is then the deviation of the output from the input. This reconstruction ignores implicit space; the output activities only depend on the representation vector

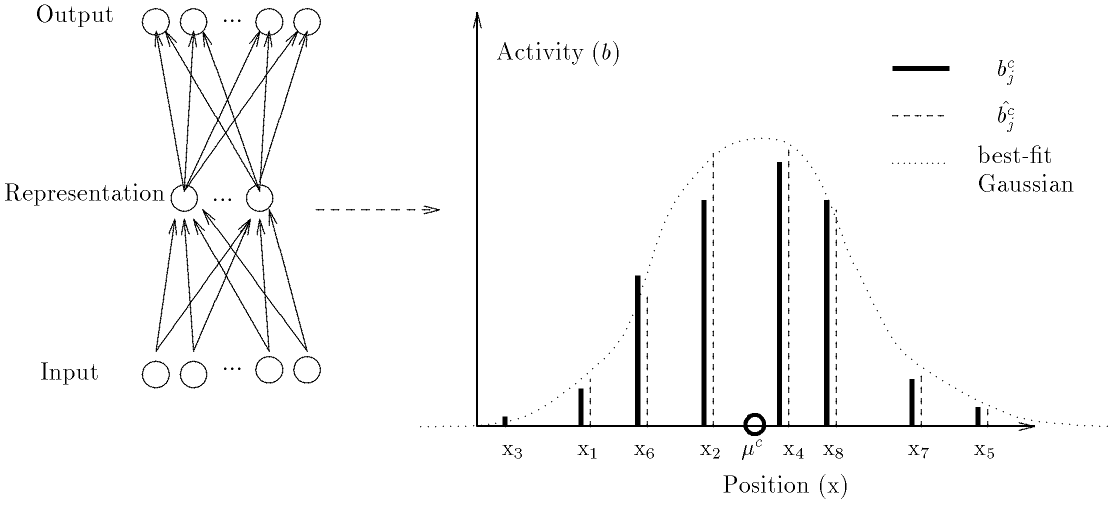



Figure 6.2: Each of the $M$ representation units in the self-supervised network has an associated position (here marked by an "x") in implicit space. This figure shows a 1D implicit space. The activity $b_j^c$ of each representation unit $j$ on case $c$ is shown by a solid line. The network fits the best Gaussian to this pattern of activity in implicit space. The predicted activity, $\hat{b}_j^c$, of unit $j$ under this Gaussian is based on the distance from $\mathbf{x_j}$ to the mean $\mu^{\mathbf{c}}$; it serves as a target for $b_j^c$.

and weights.

### 6.2.2 The communication model

The Gaussian bump acts as a prior model for the pattern of activity in implicit space; we specify the representation unit activities with respect to this prior. This algorithm therefore has a different communication model than the algorithms described in Chapters 4 and 5. Here we describe this model, and use it to derive the objective function.

The information that the sender must communicate to the receiver consists of two main parts (see Figure 6.3). The first section of the message is transmitted at the beginning of each epoch and is static with respect to each input case, while the rest depends on the particular examples. The weights $\mathcal{W}$ connecting the representation to output units, the coordinates $\{\mathbf{x_j}\}$ of the representation units in implicit space, and the standard deviation $\sigma$ of the Gaussian bump comprise the first part. The second part specifies, for each case $c$, the mean $\mu^{\mathbf{c}}$ of the bump in implicit space, and two error terms: the bump error $\mathcal{B}^c$ (the deviation from the activity in the representation layer predicted by a perfect bump), and the reconstruction error $\mathcal{R}^c$ (the deviation in predicted activity in the output layer). Given this message, the receiver can use the bump parameters ($\mu^{\mathbf{c}}$ and $\sigma$) and the representation units' implicit coordinates $\{\mathbf{x_j}\}$ to predict each representation unit's activity, and then refine these using $\mathcal{B}^c$. He can then apply the transform specified by $\mathcal{W}$ to determine the output unit activities, and finally refine these using $\mathcal{R}^c$ to arrive at the original input.

The objective function for the network is then the cost, in bits, of this message. When there are many training cases, the cost of the first part is negligible. Currently, we assume a fixed cost for specifying $\mu^{\mathbf{c}}$ on each case. We are then left with two quantities to communicate: the bump-error

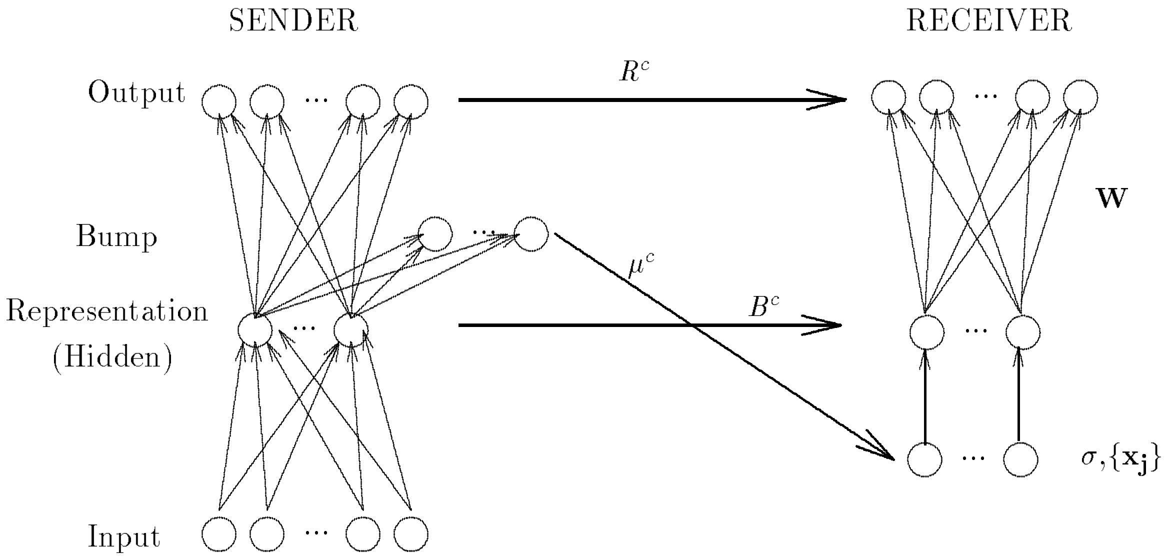



Figure 6.3: A communication model describing the information that the sender must provide in order for the receiver to fully reconstruct the data set. The sender uses a self-supervised network, and also fits a Gaussian to the pattern of activity in implicit space on each training example. The sender must first communicate the representation to output unit weights ($\mathcal{W}$), the bump standard deviation ($\sigma$), and the implicit coordinates of the representation units $\{\mathbf{x_j}\}$. For each case $c$, she then communicates the bump mean ($\mu^{\mathbf{c}}$), the bump-error ($\mathcal{B}^c$) and reconstruction-error ($\mathcal{R}^c$). The receiver can reconstruct the representation activities by computing the height under the Gaussian with parameters ($\mu^{\mathbf{c}}$,$\sigma$) at each position $\mathbf{x_j}$, and then applying $\mathcal{B}^c$. He then recovers the inputs by applying the weights $\mathcal{W}$ and $\mathcal{R}^c$. The cost of encoding each of these six terms is the full MDL objective function. In the high data limit, and ignoring the cost of encoding the mean, the sender trains the network weights and implicit coordinates of the representation units by back-propagating the derivatives of the reconstruction cost $C_{\mathcal{R}}$ from the output units, and then adding in the derivatives of the activity-cost $C_{\mathcal{A}}$ at the representation layer.

$\mathcal{B}^c$ and the reconstruction-error $\mathcal{R}^c$.

### 6.2.3 The activity cost

In this algorithm, the activity-cost $C_{\mathcal{A}}$ for each case is the bump-error. We encode the bump-error using a zero-mean Gaussian with fixed variance $V_{\mathcal{B}}$, based on the assumption that for each representation unit $j$, $\mathcal{B}_j^c$ is normally and independently distributed around the target $\hat{b}_j^c$.[2]

$$C_{\mathcal{A}} = \sum_{j=1}^{M} |b_j^c - \hat{b}_j^c|^2 / 2V_{\mathcal{B}} \tag{6.1}$$

In computing the distance from $b_j^c$ to $\hat{b}_j^c$, we assume periodic boundary conditions in order to avoid edge effects.

We compute the actual activity of a representation unit, $b_j^c$, as a normalized exponential of its total input:

$$b_j^c = exp(net_j^c) / \sum_{i=1}^{M} exp(net_i^c) \tag{6.2}$$

where $net_j^c$ is the net input into unit $j$ on case $c$. Note that a unit's actual activity is independent of its position in implicit space.

Its expected activity is its normalized value under the predicted Gaussian bump:

$$\hat{b}_j^c = \exp(-(\mathbf{x_j} - \mu^{\mathbf{c}})^2 / 2\sigma^2) / \sum_{i=1}^{M} \exp(-(\mathbf{x_i} - \mu^{\mathbf{c}})^2 / 2\sigma^2) \tag{6.3}$$

where $\sigma$ is the fixed width of the bump.

We have explored several methods for computing the mean of this bump. Simply computing the center of gravity of the representation units' positions, weighted by their activity, produces a bias towards points in the center of implicit space. Instead, on each case, a separate minimization determines $\mu^{\mathbf{c}}$; it is the position in implicit space that minimizes $C_{\mathcal{A}}$ given $\{\mathbf{x_j}, b_j^c\}$.

### 6.2.4 The reconstruction cost

We also assume that for each output unit $k$, the reconstruction-error $R_k^c$ is normally and independently distributed around the target $d_k^c$, the activity of the corresponding input unit. We encode this reconstruction-error for this algorithm using a zero-mean Gaussian with fixed variance $V_{\mathcal{R}}$. The reconstruction-cost is then:

$$C_{\mathcal{R}} = \frac{1}{2V_{\mathcal{R}}} \sum_{k,c} (y_k^c - d_k^c)^2 \tag{6.4}$$

where $y_k^c$ is the actual activity of unit $k$ for case $c$, and $d_k^c$ its desired activity.

As in the previous algorithms, the network has full inter-layer connectivity, and linear output units; the actual activity of each output unit is just the weighted sum of its inputs.

[2] This Gaussian assumption is again only an approximation to the true distribution. In this case, both $b_j^c$ and $\hat{b}_j^c$ are bounded between 0.0 and 1.0, so the error is bounded between -1.0 and 1.0, and the true distribution has a mean of 0.0 and falls off exponentially in both directions to these bounds.

Note that unlike the previous algorithms, we are not treating the activities of the representation units as stochastic binary variables. The reconstruction cost, as well as the activity cost, are not expected costs, but rather actual costs based on the training examples (see the Discussion section below). The final cost function for the network is the sum of these two costs:

$$\begin{aligned} C &= C_{\mathcal{A}} + C_{\mathcal{R}} \\ &= \sum_{j=1}^{M} (b_j^c - \hat{b}_j^c)^2 / 2V_{\mathcal{B}} + \sum_{k=1}^{N} (y_k^c - d_k^c)^2 / 2V_{\mathcal{R}} \end{aligned} \tag{6.5}$$

Both the network weights and the implicit coordinates of the representation units are adapted to minimize $C$.

## 6.3 Experimental Results

### 6.3.1 Parameters

The algorithm includes three adjustable parameters. The width of the Gaussian bump in implicit space, $\sigma$, is set in each experiment to be approximately 1/4 of the width of the space. In many learning algorithms that include a Gaussian fitting step, such as the Kohonen algorithm (Kohonen, 1982), the elastic net (Durbin and Willshaw, 1987), and the mixture-of-Gaussians, this width is initially large and is then annealed down to a minimum width during training. We have also experimented with this approach, but it did not significantly improve the algorithm's performance, so the width is fixed throughout training in the results discussed below.

The second and third parameters are the variances of the coding Gaussians for the reconstruction and activity costs. The relative values of these two terms acts as a regularization parameter, trading off the weighting of the two costs. We did not perform extensive experiments to determine optimal settings of these terms, but instead selected values within a range between the two extremes of doing straight reconstruction and ignoring the bump error, and forming the same perfect bump on each case and ignoring the input. Except when different values are given below, for these experiments, $V_{\mathcal{B}} = 1.0$ and $V_{\mathcal{R}} = 1.0$.

Two architectural parameters also play important roles in this algorithm. The first is the number of representation units. In order to accurately and robustly represent a wide range of values in implicit space, the network must contain a sufficient number of representation units to make a true population code. The second parameter is related to the first—the number of dimensions in implicit space. Clearly many more units are required to form population codes as we increase the dimensionality of implicit space. In the experiments below, we predetermine the appropriate number of dimensions for the input set; in the Discussion section we describe an extension which will allow the network to automatically determine the appropriate dimensionality.

We train the networks in these experiments using a batch conjugate gradient optimization technique, with a line-search. The results described below represent best-case performance, as the algorithm occasionally gets stuck in local minima. In these experiments, if the network contains a sufficient number of representation units, and the input is devoid of noise, then the algorithm should

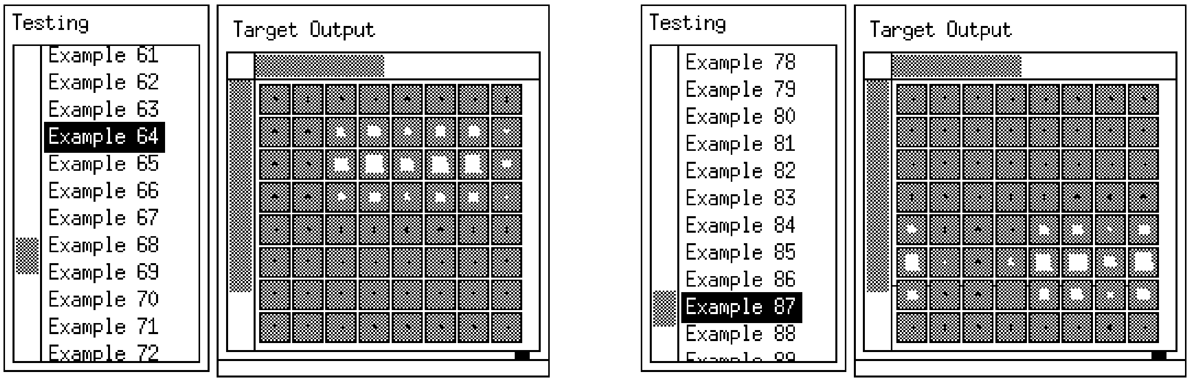



Figure 6.4: Each 8x8 real-valued input image for the first experiment contains an instance of a dipole. The dipole has four continuous degrees-of-freedom: $(x, y)$-position, orientation, and size. This figure shows two sample images from the test set. The image on the right shows that the input space is represented using wraparound.

be able to force the cost function to zero (since we are ignoring the cost of communicating the bump means). This makes it relatively easy to determine when a solution is a local minimum.

### 6.3.2 Experiment 1: Learning to represent position

The stimuli used in the first two experiments are very simple. Each 8x8 real-valued input image contains a rigid object, which is composed of two Gaussian blobs of intensity at its ends (see Figure 6.4). The image is generated by randomly choosing a point between 0.0 and 1.0 to be the center of the object. The two ends of the object are then a fixed distance from this center (each is approximately 0.2 units displaced from the center, where the entire image is 1.0 units wide). Each end is then composed of the difference of two Gaussian blobs of intensity: one of standard deviation 0.1 units, and a second with a standard deviation of 0.25 units, which acts to sharpen up the edges of the object.

This simple object has four degrees of freedom, as each instantiation has a unique $(x, y)$-position, orientation (within a 180 degree range), and size (based on the spacing between the ends). These four *instantiation parameters* of the object describe the variability due to seeing the same rigid 2D object from different viewpoints in the fronto-parallel plane. This level of complexity is sufficient for initial experiments; as the later experiments demonstrate, the method is capable of learning from more complex images as well. In order to avoid edge effects, the input space was represented using wraparound, i.e., any aspect of the object that was located off one edge of the image would appear on the opposite edge.

In the first experiment, only the $(x, y)$-position of the object is varied from image to image. The training set consists of 400 examples of this shape in random positions; we test generalization with a test set of 100 examples, located at the gridpoints of the 10x10 grid covering the space of possible positions. The network begins with random weights, and each of 100 representation units has random 2D implicit coordinates. The system converges to a minimum of the objective function after 23 epochs.

We can estimate the number of bits in the training set of this algorithm. Each image is generated

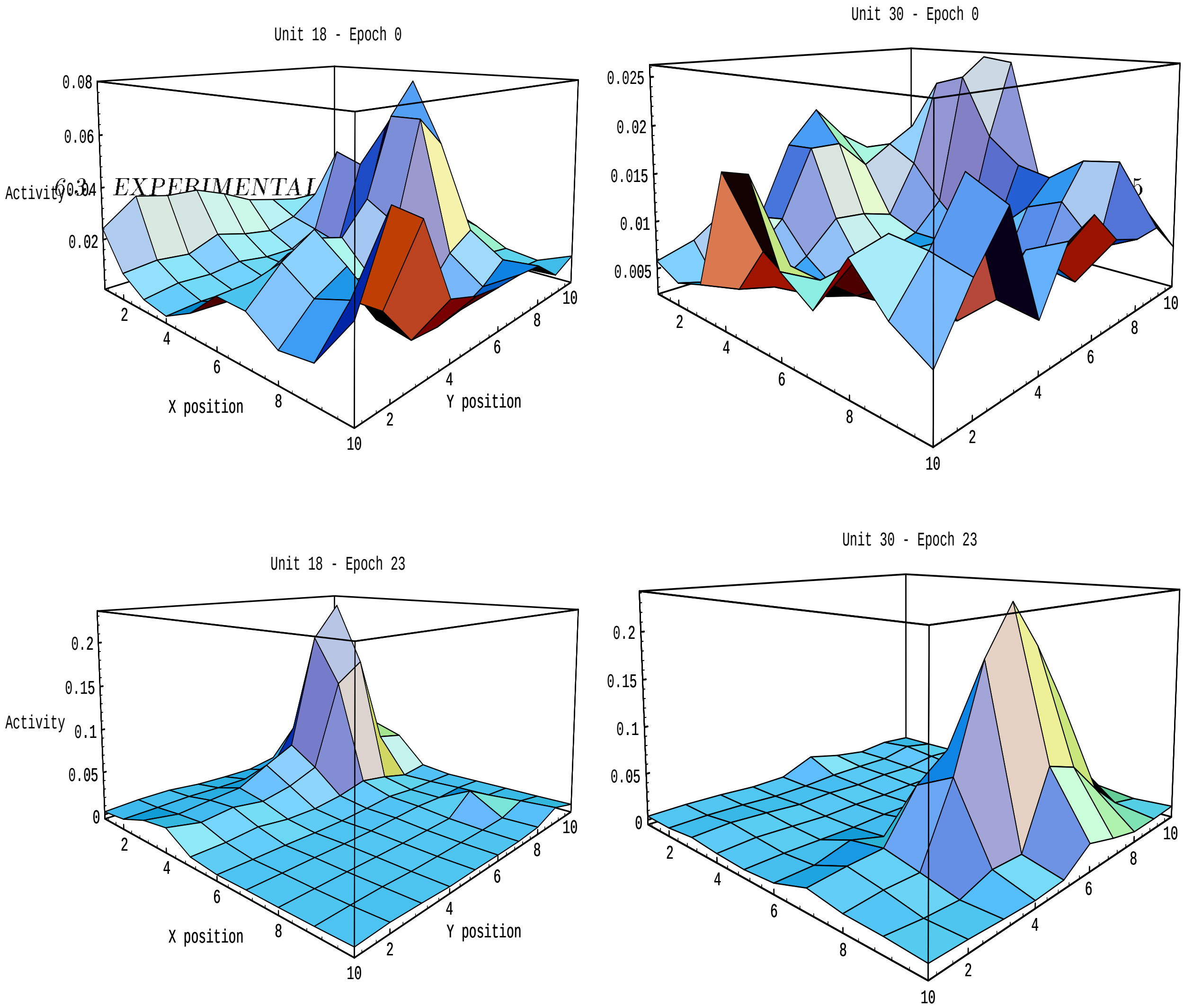



Figure 6.5: This figure shows the receptive field in implicit space for two representation units. Here the 2 dimensions in implicit space correspond to $x$ and $y$ positions. The top panel shows that before learning, the units respond randomly to 100 different test patterns, generated by positioning a shape in the image at each point in a 10x10 grid. The lower panel shows that after learning, the units respond to objects in a particular position, and their activity level falls off smoothly as the object position moves away from the center of the learned receptive field.

by randomly selecting an $x$ and $y$ coordinate for the shape, so if each coordinate is represented to an accuracy of 4 bits, since we know that the image contains an instantiation of a known shape, then the image contains 8 bits of information. In this algorithm, we are assuming that these 8 bits are the fixed description length required to communicate the implicit coordinates of the bump mean. The generalization length (the mean description length (see Equation 6.5) of the test set) on this experiment is 0.52 bits, indicating that the algorithm was able virtually to eliminate the bump and reconstruction errors.

The next two figures demonstrate that the system develops a population code representation of the images, where the dimensions of the implicit space correspond to the underlying dimensions of variability in the input set, which here relate to the position of the object. Figure 6.5 shows that each representation unit develops a *receptive field*, as it responds to objects in a limited neighborhood of the underlying constraint space that corresponds to its learned position in implicit space.

Figure 6.6 shows that the algorithm learns the underlying constraint surface in this dataset. Each input can be characterized by the $(x, y)$-coordinates of the object in image space, and the means in implicit space comes to correspond to these coordinates. This figure shows that implicit space forms a relatively smooth, stable map of this generating surface, i.e., a small change in the

$(x, y)$-coordinates of the object produces a corresponding small change in the mean's coordinates in implicit space.

### 6.3.3 Experiment 2: Learning a three-dimensional constraint surface

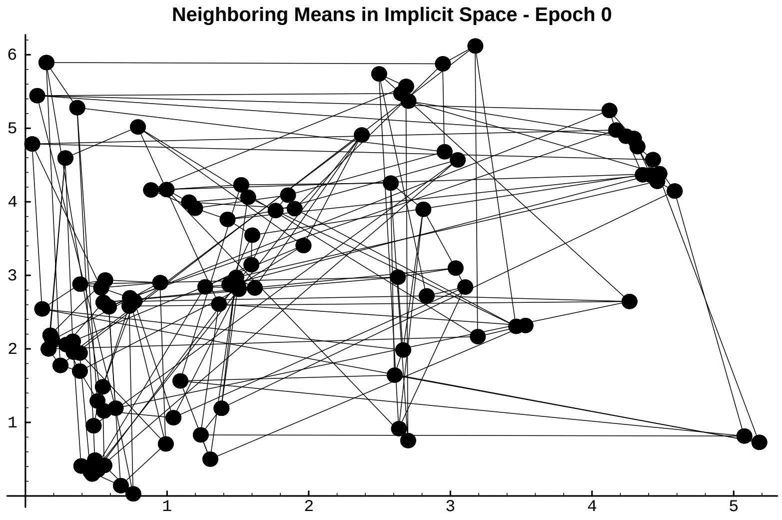
In the second experiment, we also vary the orientation of the shape. The training set contains 1000 images of the same object with 3 random instantiation parameters, i.e., its position and orientation are drawn randomly from the range of $(x, y)$-coordinates and 180 degrees of orientation. The test set contains 512 examples, made up of all gridpoints in an evenly-spaced grid that divides each of the three underlying dimensions into 8 intervals. Figure 6.7 shows two images from this test set.

We give each representation unit three implicit coordinates, and also increase the number of representation units to 225. Since this network has a larger ratio of representation to input units, we increase the bump variance ($V_{\mathcal{B}} = 2.5$) in order to maintain a balance between the two costs. The network converges after 600 epochs of training on the 1000 images. The generalization length is 1.16 bits per image. The representation unit activities again form a population code that allow the input to be accurately reconstructed. Figure 6.8 shows the pattern of activity in the trained network on two images from the test set.

The three dimensions of the implicit space correspond to a recoding of the object instantiation parameters, such that smooth changes in the object's parameters produce similar changes in the implicit space codes. This is difficult to display in the 3D implicit space. Instead, Figure 6.9 shows the outgoing weights for 6 units that end up nearby in implicit space.

While the algorithm is capable of finding the true generator for this task, it has a tendency to get stuck in local minima. These minima typically correspond to piecewise smooth solutions, i.e., local neighborhoods, or patches in implicit space correspond to patches of the constraint surface, but the solution is not smooth across the entire surface. This type of solution is found particularly when we reduce the number of representation units. With fewer than 150 units, the algorithm is unable to form a bump for each input, and the reconstruction-error also increases. Increasing the number of units to more than 225, however, virtually solves the local-minimum problem; as in the algorithms described earlier in the thesis, the search for a good representation of the input set is aided by having an excess of representation units, since the algorithm can use the large pool of units to initially find representatives of the various inputs, and then effectively remove the excess units.

The problem of local minima is common in unsupervised learning, and topographic-map formation algorithms are no exception. We discuss local minima further in the Discussion section below. In this experiment, an additional possible explanation for the local-minima problem is the *curse of dimensionality*, which states that the number of samples required to cover a space grows exponentially with the dimensionality of the space. Thus the random samples in the training set may be over-sampling some areas of space relative to others, and those relatively dense areas are smoothly represented in implicit space. This problem has consequences for the scaling properties of the algorithm; we return to this point in the Discussion section.



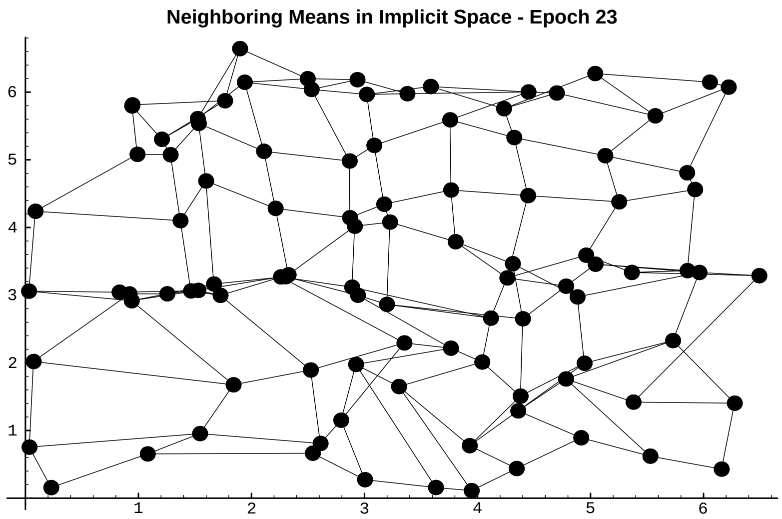




Figure 6.6: This figure shows the implicit positions of the bump means for the test set before and after training. In the 100 testing examples, the object is located at the gridpoints of a 10x10 grid covering the space of possible positions. In this figure, lines connect the means between a given test image and its 4 neighbors on this grid. Note that implicit space has a toroidal shape, i.e., the points at $2\pi$ radians are neighbors of the points at 0 radians. The lines connecting these wraparound neighbors have been left off of this figure to improve its clarity.

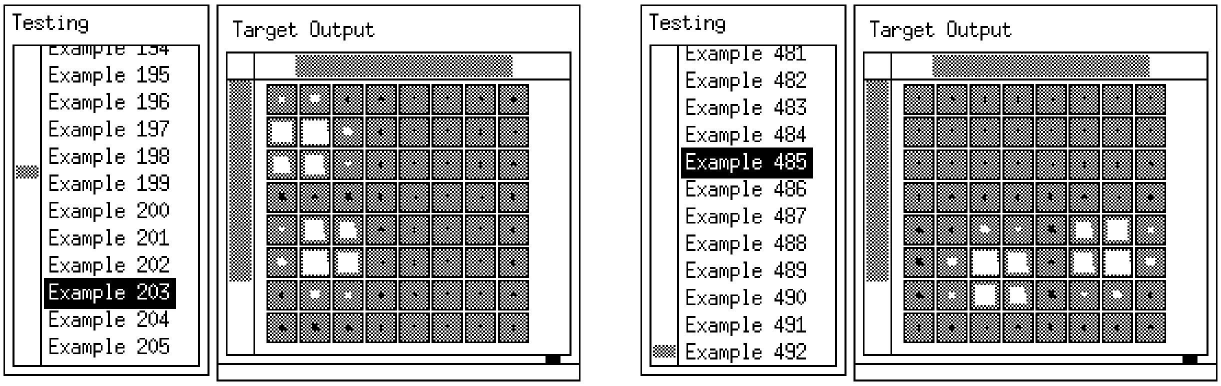



Figure 6.7: Two sample images from the test set for the second experiment. Each of the 1000 images in the training set contains an instantiation of the object with random $(x, y)$-coordinates and a random orientation. The test set is an ordered sampling of these three object instantiation parameters, which form the underlying constraint surface in this dataset.

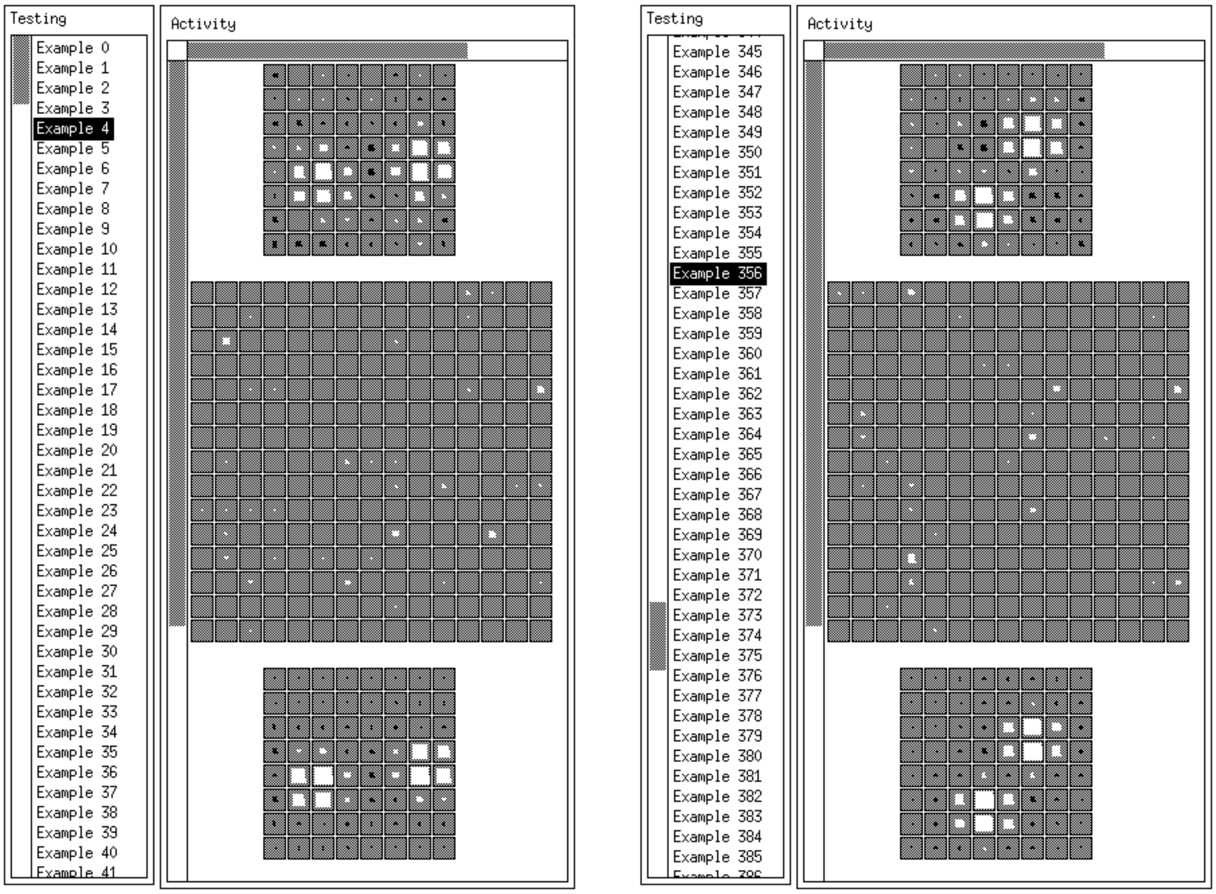



Figure 6.8: This figure shows the patterns of activity in the input, representation, and output units for two test examples from the second experiment. Note that the representation units' activity levels are not decipherable as a Gaussian bump here. The bump depends on the implicit coordinates of each representation unit, which are not shown here. This emphasizes the point that a population code involves 3 spaces: an input space, a hidden space, and an implicit space.

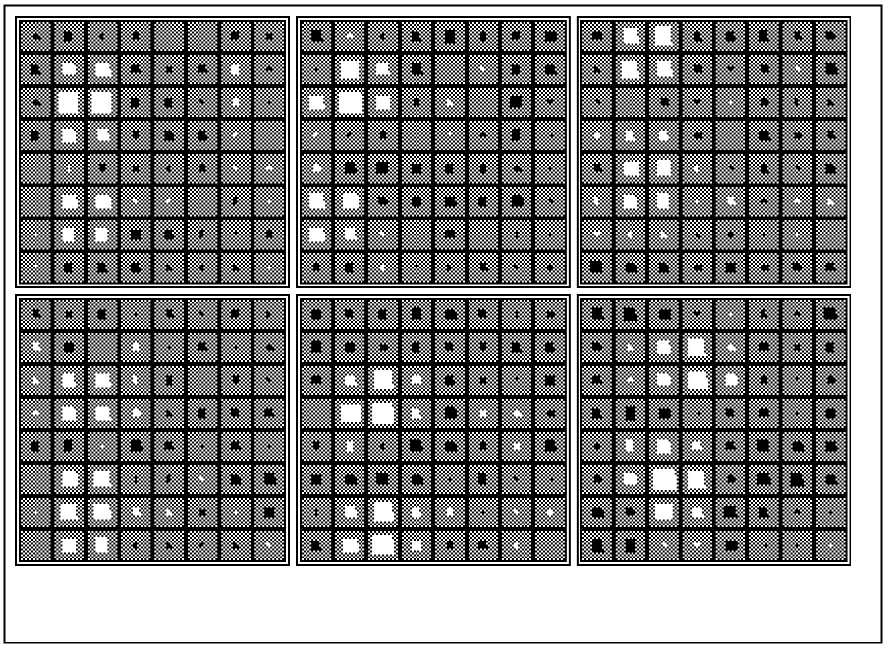


Figure 6.9: This figure shows the outgoing weights for six representation units in the trained network from the second experiment. These six units have similar implicit coordinates, and their contributions to the reconstruction are also similar, indicating that the network has learned a smooth, stable correspondence between implicit space and the constraint surface in this dataset.

### 6.3.4 Experiment 3: Learning a discontinuous constraint surface

A third experiment employs a training set where each image contains either a horizontal or vertical bar, in some random position (see Figure 6.10). The bar has a width of 0.15 units, and a length of 0.6 units, where the 8x8 image is 1.0 units square. The pixel intensities are determined by a negative exponential function of their squared distance to the bar. The training set contains 200 examples of each bar, and the test set contains 100 examples of each, evenly-spaced at 10 locations along both dimensions of the underlying constraint surface, the $(x, y)$-position of the shape.

This task is a what/where problem: the underlying constraint surface has the two continuous dimensions of position, but it also has a binary value that describes which object (horizontal or vertical) is in the image. Even though we only give each of the 100 representation units 2 implicit coordinates in this experiment, they are able to discover a representation of all 3 of these underlying dimensions. The algorithm requires 112 epochs to reduce the generalization length to 1.4 bits. This generalization length is nearly 3 times that of Experiment 1, where the network had the same number of representation units trained on a single shape in various positions. The decrease in representational quality can be attributed to this additional "what", or identity dimension of the training set.

After training, we find that one set of representation units has moved to one corner of implicit space, and represent the position of instances of one shape, while the other group has moved to an opposite corner and represent the position of the other shape (see Figure 6.11). The network

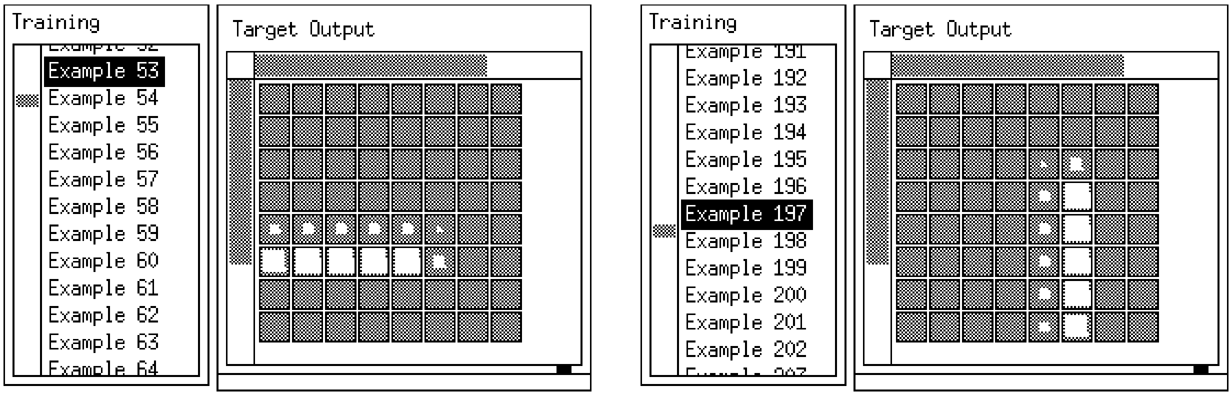



Figure 6.10: This figure shows one instance each of the horizontal and vertical bar images from the third experiment.

sometimes finds solutions where rather than identity being the primary dimension in implicit space, the units instead cluster according to the shape location, and the representation of identity is patchy within this ordering. This second solution has a higher generalization length (1.8 bits).

A Kohonen network (Kohonen, 1982) typically finds this second type of representation in which position is the primary implicit dimension. We have run a Kohonen network with 100 output units on this horizontal/vertical task. After extensive parameter-tweaking, the network learns a patchy topographic map, where the underlying constraint surface is captured in a local fashion by the map (see Figure 6.12). We were unable to learn the other representation using the Kohonen algorithm; the difficulty in learning this sort of mapping can likely be attributed to the fact that the implicit space topology remains fixed while the network learns to associate points in implicit space with points in input space. The fact that the representation units *learn* their implicit coordinates in our algorithm allows much more flexibility than any system in which these coordinates are fixed in advance.

A related problem to this what/where task is the development of ocular dominance and retinotopic maps in striate visual cortex. Inputs from the two eyes' retinae are mapped in V1 such that neighboring retinal positions innervate nearby neurons in V1, and these nearby neurons also tend to respond to input from one eye or the other. In this mapping, location is the primary dimension, and the binary dimension, ocularity, is secondary; the resulting map resembles the patchy structure shown in Figure 6.12. Obermayer, Ritter, and Schulten (1991) showed how a Kohonen algorithm trained on this task could develop a similar map, and analyzed the dependence of the map structure on certain key underlying input parameters.

## 6.4 Discussion and current directions

We have shown how our MDL framework can be used to develop non-factorial, redundant representations from a set of input images. Instead of communicating each representation unit activity independently, we communicate the location of a Gaussian bump in a low-dimensional implicit space. If the representation units are appropriately tuned in this space their activities can then be inferred from the bump location. When the underlying dimensions of variability in a set of input images

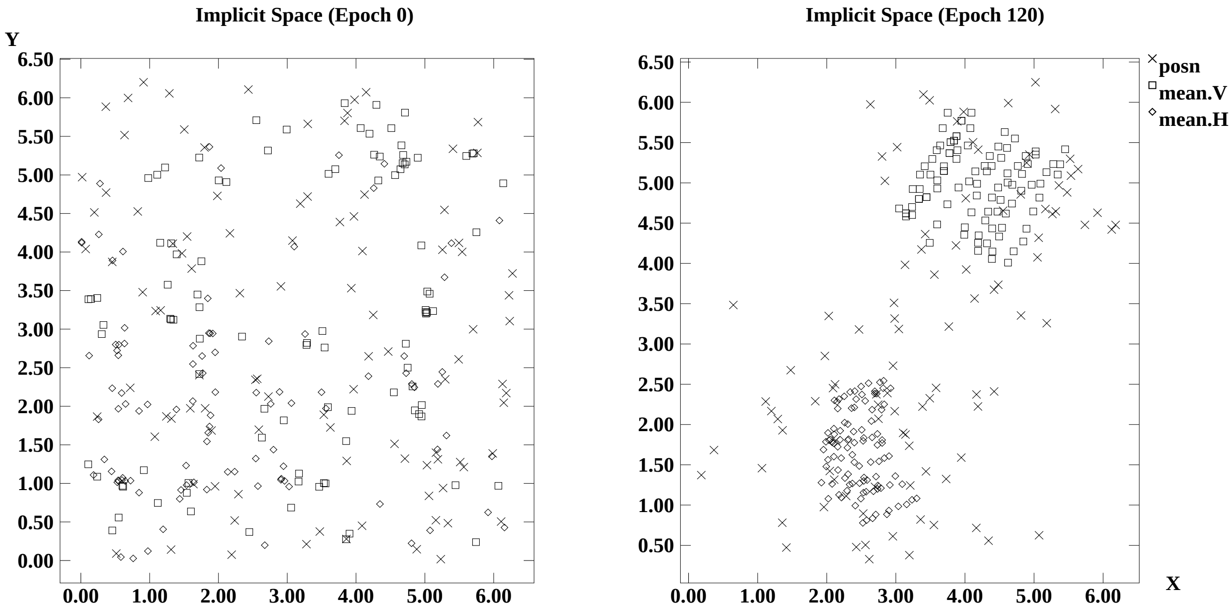



Figure 6.11: This figure shows the positions of the representation units and the means in the 2D implicit space before and after training on the horizontal/vertical task. In each plot, an "x" marks the implicit position of a representation unit; a box marks the implicit mean of the bump formed for an input image containing a vertical bar; and a diamond marks the mean of a horizontal bar image. The network has learned a topology where the means in the top right of the second plot all correspond to images containing vertical bars, while the other set correspond to horizontal bar images. The different shapes are thus separated in implicit space. Note that some representation units are far from all the means; these units do not play a role in the coding of the input, and are free to be recruited for other types of input cases.

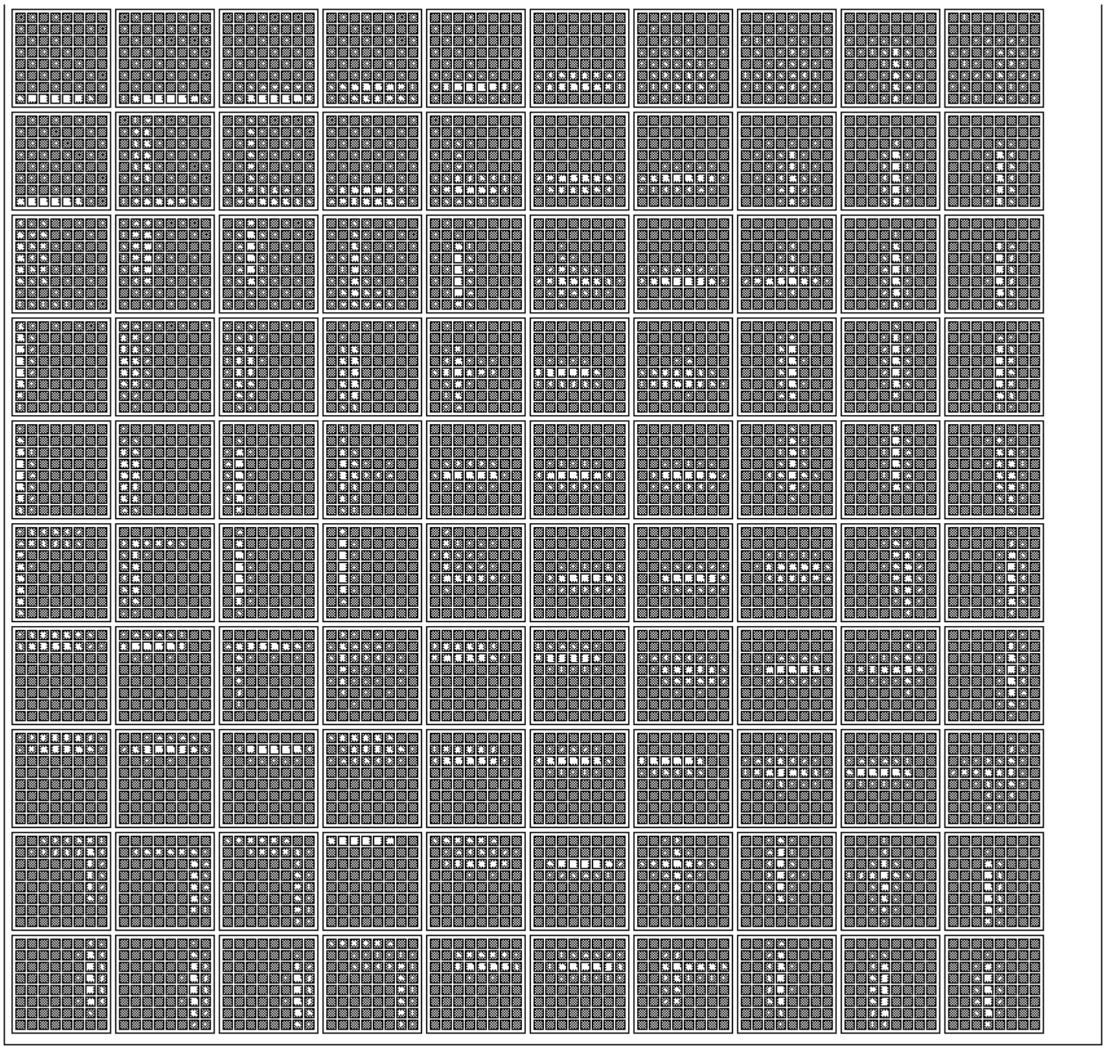


Figure 6.12: This figure shows the incoming weights for the 10x10 map of output units in a Kohonen network trained on the horizontal/vertical task of Experiment 3. The network was trained using gradient descent, with a small learning rate ($\alpha = 0.0001$); a broad Gaussian neighborhood function ($\sigma = 10.0$, the entire width of the map), and a slow annealing schedule (decays with $1/(1+0.00001*t)$, where $t$ is the number of training epochs) in order to avoid local minima. The resulting map displays local topographic structure, as nearby units in the output map have similar incoming weights, and the underlying constraint surface—the $(x, y)$-coordinates of the bar—is smoothly represented within a local patch. Note that a local patch of units responds to horizontal bars, and a neighboring patch responds to vertical bars, and the units inbetween respond to both.

are the instantiation parameters of an object, this implicit space comes to correspond to these parameters. Since the implicit coordinates of the representation units are also learned, the network develops separate population codes when presented with different objects.

### 6.4.1 Comparison to standard methods

This new algorithm bears some similarities to several earlier algorithms. In the experiments presented above, each representation unit learns to act as a Radial Basis Function (RBF) unit (see Section 1.3.1). Unlike standard RBFs, however, here the RBF activity serves as a target for the activity levels, and is determined by distance in a space other than the input space.

Our algorithm is more similar to topographic map formation algorithms, of which the Kohonen algorithm is the most well-known example. These two algorithms share the aim of creating a map where the constructed, or implicit, space corresponds to the underlying constraint space of the inputs. They both create this structure primarily by indirect local learning. The Kohonen algorithm finds a winner for each training case and then moves the weights of nearby units towards those of the winner. Our algorithm works in a similar manner, as the network tends to associate a few units with each image, and those units that account for similar images get assigned similar implicit coordinates. This procedure operates locally, however, so there is no strong force exerting global order on the implicit space-constraint surface correspondence.

Several important differences exist between the two algorithms. First, while both algorithms assume that each input case corresponds to a single location in the underlying constraint surface, the Kohonen algorithm uses the most active unit to identify this location, while our algorithm locates the peak of the Gaussian bump based on the pattern of activity in the representation layer. His method is thus limited by the number of units in the map, while our method possesses the hyperacuity properties of population codes.

The Kohonen algorithm is a computational approximation to the Willshaw and von der Malsburg (1976) algorithm (see Section 3.3.3), where lateral interactions create a bubble of activity (just like in a Gaussian bump) around the most active unit on each input. The Kohonen algorithm then, in effect, assumes that the neighbors of the winner have non-zero activity levels, so these neighbors also have their weights changed. But the algorithm tries to make each neighbor be more like the whole input vector, rather than changing the weights, commensurate with the bump, to capture the aspects of the input that are not accounted for by the other active hidden units. Our algorithm adopts this latter approach, enabling it to form componential representations of the input (see Section 6.4.2 below).

A related difference between the algorithms is that since the Kohonen algorithm moves neighbors' weights towards the input patterns, the population-code is in effect formed in input space.[3] Population coding in a space other than the input enables our networks to extract nonlinear higher-order properties of the inputs.

[3] In Luttrell's (1990) interpretation of Kohonen's algorithm in terms of a communication model, the algorithm minimizes the expected distortion between the decoding of an encoded input with added noise, and the input itself. This minimization produces the Kohonen learning rule, moving neighbors of the winner towards the input patterns, when the encoding is calculated by ignoring the added noise and finding the point that the decoding maps closest to the input (see Section 3.3.3).

Finally, the topology of implicit space is learned in our algorithm rather than predetermined. Learning this topology lends additional flexibility to our algorithm. While the Kohonen and elastic net algorithms employ fixed topologies, a recent variation of Kohonen's algorithm (Martinetz and Schulten, 1991) learns the implicit coordinates (still in input space), and also allows different parts of implicit space to have different dimensionalities.

While our algorithm has proven successful on the experiments described above, it has several limitations. We are currently addressing some of these limitations, as we extend this work in a number of directions.

### 6.4.2 Fitting multiple bumps

An important underlying assumption in the algorithm as described here is that by fitting a single Gaussian to the pattern of activity in implicit space, we are presuming that each input corresponds to a point on the underlying constraint surface. For example, in the experiments, each image contained only one instance of the object.

One straightforward extension would be to fit multiple bumps to each input. For example, we might divide implicit space into local patches, and fit a bump in each patch.[4] This extension would allow the algorithm to learn when the input is comprised of multiple components drawn from the underlying constraint surface.

This multiple-bump extension would make this population-code algorithm similar to the CVQ algorithm described in the previous chapter, in that different components of the underlying constraint surface are represented in different groups of representation units, and these groups cooperate to reconstruct the input. The important difference, however, is that here we are not communicating the single active unit in each group, but instead the single point in implicit space based on the pattern of activity in the group.

A sensible initial test for this multi-bump formulation would be the splines dataset described in the previous chapter. There we found that the CVQ algorithm often formed a representation of a component of a spline by interpolating between two units within a VQ. Two partially active units did not create a cheap encoding, as this introduced variance into the reconstruction. In the algorithm from this chapter, however, we expect units to be partially active, as the combination of active units defines any particular point. Hence we would expect this algorithm to form cheap encodings of the splines.[5]

This multi-bump approach suggests one method of confronting the curse of dimensionality with respect to the population-code representation of high-dimensional constraint surfaces. In the experiments we describe above, we have constructed implicit space as a single $n$-dimensional space. An alternative layout for the space is to separate the dimensions, so that we fit a 1-dimensional Gaussian bump in $n$ different implicit spaces. Then the means and bump-errors are communicated independently.[6] Using several lower-dimensional implicit spaces could reduce the number of

[4] This would necessitate including the ability to fit a "null" bump.

[5] The multi-bump formulation would also be useful in developing a third type of representation for the horizontal/vertical task of Experiment 3. One bump could represent object identity, and a second bump could represent where, and the two would be combined (via some intermediate units) to reconstruct the input.

[6] Note that the use of spherical Gaussians in the algorithm described here presumes that the dimensions in implicit

representation units required to represent the constraint surface.

The alternative implicit space formulation points up some similarities to the model proposed by Saund (1989). His algorithm also trained patterns of activity in separate sets of hidden units in a self-supervised network to form one-dimensional Gaussian bumps, where the center of the bump was intended to correspond to the position in an underlying dimension of the inputs. His objective function contained a reconstruction-error term and a bump-error term, but this latter term was the deviation between a smoothed version of the hidden activities and the activities themselves. In addition to the objective functions being quite different in the two algorithms, another crucial difference exists: in his algorithm, the implicit space was statically determined by the ordering of the hidden units, while units in our model learn their implicit coordinates.

### 6.4.3 Automatic selection of implicit dimensionality

A second limitation of the current algorithm is the need to predetermine the number of dimensions in implicit space; we are currently working on an extension that will allow the learning algorithm to determine for itself the appropriate dimensionality. We start with many dimensions but include the cost of specifying the bump mean ($\mu^{\mathbf{c}}$) in the description length. This depends on how many implicit coordinates are used. If all of the representation units have the same value for one of the implicit coordinates, it costs nothing to communicate that value for each bump.

If we assume that the means themselves form a Gaussian distribution across the set of bumps, then the description length for the means is given by the following equation (see Section 2.3.2).

$$|\mathcal{L}^*(\mu)| \approx N \sum_d \log[Var(\mu_d)/\epsilon] \tag{6.6}$$

where $N$ is the number of training cases, $\epsilon$ is the tolerance to which the means are encoded, and the mean along each dimension $d$ is encoded independently.

The cost of an implicit coordinate thus depends on the ratio between its variance (over all the different bumps) and the accuracy with which it must be communicated. We can simply add this cost to the reconstruction and bump errors. The network can then save bits by reducing the variance for unneeded coordinates. This creates a smooth search space for determining how many implicit coordinates are really needed.

### 6.4.4 Higher-order networks

While the two algorithms derived from the framework described in Chapters 4 and 5 require an approximation in order to generalize to more complex networks, this algorithm lacks their stochastic nature, and thus readily extends to deeper networks. We can devise a stochastic version of this algorithm in which we interpret the pattern of activity in the representation units as a probability distribution across possible implicit space locations. Under this interpretation, however, the representation returns to one of a set of possible locations, and loses the hyperacuity properties of the

space are independent. Activity along each dimension nevertheless affects the expected activity of every unit, whereas that would not be true in the new scheme that devotes separate units to each dimension.

deterministic version described here.

The method described in this chapter can thus easily be applied to networks with multiple hidden layers, where the implicit space is constructed at the last hidden layer before the output and derivatives are then backpropagated; this allows the implicit space to correspond to arbitrarily high-order input properties. This makes the algorithm potentially useful in many vision problems, where the relevant underlying variables correspond to higher-order non-linear properties of the input.

## Chapter 7

# General Discussion

In this final chapter, we summarize the main contributions of the thesis, and close with suggestions of possible directions for future work related to the thesis.

## 7.1 Primary contributions

Our primary goal in unsupervised learning is to formulate algorithms that discover representations which uncover the true causes of datasets drawn from a variety of domains. Ideally, the algorithms would also have some common basis so that the best algorithm for a given dataset could be automatically selected.

Previous approaches in unsupervised learning are generally limited in the kinds of structure, or causes, that they uncover. For example, algorithms based on Principal Components Analysis, or more generally projection-method approaches (see Chapter 2), work best when the data distribution is ellipsoidal, so that the projection axes can correspond to the axes of the ellipsoid. Kernel methods are more appropriate when the data consists of isotropically distributed clusters, which would tend to be obscured by PCA and related methods.

Each of these algorithms have an inherent implicit assumption as to the types of structure that will underlie the data, and that assumed structure is reflected in the representations produced by the algorithm. Choosing the appropriate coding method therefore requires prior knowledge about the input distribution.

The approach taken in this thesis is novel in that by casting the problem in terms of a communication model, we are forced to make assumptions about the structure explicit, since they determine the coding scheme for the representations produced by the network, as well as determining the appropriate activation functions and network architecture. Then by formulating the problem in Minimum Description Length terms, we obtain a common yardstick with which we can compare the representations produced by different algorithms generated from within the framework.

The first contribution of the thesis then is this novel framework for unsupervised learning procedures (Chapter 2). We start by making constraining assumptions about the types of structure that we expect in the domain of interest. We then formulate a scheme for encoding representations that contain these types of structure, as well as appropriate network architectures and activation functions. This procedure then results in a new learning algorithm that will tend to extract representations with the desired structure from datasets.

We illustrated this procedure by applying it to generate three different learning algorithms. The first algorithm (Chapter 4) employed a simple type of structure, where the data was assumed to consist of a wide vocabulary of features, only a small subset of which were present in any given example. We showed that this algorithm successfully uncovered the true causes of a dataset where each image contained a few horizontal and/or vertical lines. The network learned to associate a single representation unit to each potential line in the dataset, so that a few active units could succinctly describe any image. This algorithm found a good representation for a second dataset, where each input consisted of two components, each of which contained one of 13 possible patterns. Each of 26 representation units came to correspond to one of the 13 patterns in one of the two components.

The second algorithm (Chapter 5) involved a slightly more structured data description, where the data was assumed to consist of multiple feature categories, which in turn contained a set of discrete alternatives. This algorithm found a more succinct description of the second dataset described in the previous paragraph, as each component became a feature category, and the 13 patterns formed the set of discrete alternatives within each category. This representation was less redundant than that found by the first algorithm. This second algorithm also found non-redundant representations for a dataset containing spline images. We showed that the algorithm was capable of finding the best representations (in MDL terms) given the architecture. With four categories, the network found one decomposition of the splines, while with five categories, it found another, slightly more efficient decomposition.

The third algorithm (Chapter 6) assumed that the structure lay in a low-dimensional underlying space of variability in the input set, where each example was considered a point in this continuous space. This algorithm encouraged the formation of population-code representations, where the dimensions of the population code came to correspond to the continuous underlying dimensions of input variability. We showed that this algorithm discovered the underlying dimensions of viewpoint variability in a set of images, as well as learning distinct representations when the input images contain different shapes.

In addition to the general framework and the specific algorithms, the thesis contains other novel contributions. First, we show that employing stochastic units in the network can lead to more compact, efficient representations while also aiding search, and we present the cost of communicating using these noisy representations. Also, we demonstrate in Chapter 6 that the MDL approach is capable of forming redundant representations, even though this seems to run counter to the ideas of MDL. In addition, we show how population-code and topographic-map formation algorithms may be understood in terms of constructing topologies in a space other than the input. This new viewpoint may lead to new insights about the existing algorithms, as well as novel algorithms.

Finally, the three algorithms we derived from the framework are just three examples. Further applications of this framework should lead to a range of novel algorithms that will uncover true causes of other datasets.

## 7.2 More directions for future work

In each of the three chapters describing new algorithms derived from the MDL framework, we described the primary directions in which we would like to extend the particular algorithm. In this section we describe more general directions for future work.

### 7.2.1 Include the model cost

In the communication model underlying the MDL framework, as set out in Chapter 1 (Section 1.4.1), the message that allows the receiver to fully reconstruct the input set consists of three parts: (1) the values of the weights ($\mathcal{W}$) that connect the representation units to the output units; (2) the activity levels of the representation units on each training case ($\mathcal{A}$); and (3) the error in the prediction made by the representation units, or the mismatch between the actual output and the input on each case ($\mathcal{R}$). In the new algorithms introduced in this thesis, the cost functions include the cost of encoding the last two of these terms, as we assume that in the high data limit, the cost of encoding $\mathcal{W}$ is negligible relative to these other two costs.

By ignoring the cost of the model, however, we may be missing some important structure that underlies the input set. Nowlan and Hinton (1992), and Hinton and van Camp (1993) show that MDL may also be applied to find structure in a supervised learning task. In this latter paper, the cost function is the sum of two encoding costs: the weights of the network, and the misfit between the actual and desired output of the network. On a sample problem, coding the weights using an adaptive mixture-of-Gaussians prior allowed the network to find three sharp clusters for the weights. Discovering this structure avoided over-fitting the data for this problem, as the network was able to fit a complicated nonlinear model where the number of training cases was less than the dimensionality of the input vector.

One natural extension would therefore be to include the cost of encoding the weights in the cost functions in the thesis. We can see where including this cost term could have beneficial effects on the learning of some of the datasets. For example, in Chapter 5, when we applied the CVQ algorithm to the spline images, all the units in a particular VQ learned to ignore inputs from most of the image except for a narrow band. If we coded the weights using separate priors for each VQ, then the units within the VQ would be encouraged to have similar weights, and the network might find this solution more readily.

### 7.2.2 Test utility of discovered representations

Another interesting direction for future work would be to apply the algorithms defined in the thesis in a more task-driven manner, in order to test the utility of the discovered representations. We have shown that the algorithms are capable of extracting the underlying structure in synthesized datasets. In addition to extending these algorithms to a wider range of data, we could also extend them to show that they are useful in tasks other than compressing the data.

These discovered representations may be useful in two different ways. First, we would like to show that these representations can aid further processing and learning. One natural way to do this is to let the activities in the representation layer of a network trained by one of these MDL algorithms serve as input to a supervised learning system. Intrator (1993), for example, has shown that combining an unsupervised learning procedure with a supervised one can improve generalization. Similarly, extracting structure in the data by using an MDL network as a pre-processor should lead to a faster and more efficient supervised network versus a network that operates directly on the input data.

A second method of using these representations is for classification. We have shown that the algorithms explored here can train a network to embody the underlying structure in data drawn from a common domain. One could use this trained network then to classify a novel example as to whether it is from that same domain. For a test case that does not share the learned structure, the network would produce a very high description length, while one that does should have approximately the same description length as a training example.

This method could be generalized to perform multiple-class discrimination. Separate networks could be trained on examples from different classes, one class per network. We can then define a classification criterion that estimates which of these trained networks produces a description length for a novel test example closest to those of its training examples.

### 7.2.3 Extract higher-order features

Finally, another direction for future work would be to apply the methods described in the thesis to more complicated examples, where the true causes may be higher-order functions of the input. This extension would entail adding more layers to the networks, so that the representations could correspond to something other than subsets of the input patterns. All of the algorithms described in the thesis readily generalize to networks with additional layers between the input and representation layers. Adding layers between the representation and output, however, is problematic where we use noisy representations (e.g., the algorithms described in Chapters 4 and 5). Propagating probability distributions through non-linear layers of the network would necessitate the application of either Monte Carlo methods or approximations. Also, if we incorporate the model-cost as described above, then the cost of these added weights must also be considered.

While the algorithms can thus be extended to extract higher-order features of the input, more extensive developments will be required in order to model real-world inputs. Simply adding more layers to a network will clearly not produce a representation of the underlying causes of complex data such as the medical images and jazz solos introduced way back in the beginning of this thesis. We need to develop new and more elaborate definitions of representation classes—which will translate into additional structure in the network representation—in order to begin addressing more realistic data. For example, the representation should include a notion of levels of abstraction, so that musical notes, phrases, chord progressions and whole pieces could be hierarchically represented. These remain as the larger goals for the endeavor described in the thesis.

# Bibliography


Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions AC-19*, pages 716–723.

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

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

Barlow, H. (1961). The coding of sensory messages. In *Current Problems in Animal Behavior*, pages 331–360. Cambridge U. Press, Cambridge.

Barlow, H. (1983). Understanding natural vision. In Braddick, O. and Sleigh, A., editors, *Physical and biological processing of images*, pages 2–14. Springer-Verlag, Berlin.

Barlow, H., Kaushal, T., and Mitchison, G. (1989). Finding minimum entropy codes. *Neural Computation*, 1(3):412–423.

Barlow, H. B. (1989). Unsupervised learning. *Neural Computation*, 1(3):295–311.

Barlow, H. B. and Földiák, P. (1989). Adaptation and decorrelation in the cortex. In Durbin, R. M., Miall, C., and Mitchison, G. J., editors, *The Computing Neuron*, chapter 4, pages 54–72. Addison-Wesley, Wokingham.

Baum, E. B. and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1(1):151–160.

Becker, S. (1989). Unsupervised learning procedures for neural networks. *International Journal of Neural Systems*, 2(1):17–33.

Becker, S. and Le Cun, Y. (1989). Improving the convergence of back-propagation learning with second order methods. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, Pittsburgh 1988. Morgan Kaufmann, San Mateo.

Bienenstock, E. L., Cooper, L. N., and Munro, P. W. (1982). Theory for the development of neuron selectivity; orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2:32–48.

Bridle, J. S. (1990). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 211–217, San Mateo, CA. Morgan Kaufmann.

Churchland, P. S. and Sejnowski, T. J. (1992). *The Computational Brain*. MIT Press, Cambridge, MA.

Cottrell, G., Munro, P., and Zipser, D. (1989). Image compression by back-propagation: An example of extensional programming. In Sharkey, N., editor, *Models of cognition: A review of cognitive science*, pages 208–240. Ablex, Norwood, NJ.

DeJong, G. F. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 2:145–176.

DeMers, D. and Cottrell, G. (1993). Non-linear dimensionality reduction. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 580–587. Morgan Kaufmann, San Mateo, CA.

Durbin, R. and Willshaw, D. (1987). An analogue approach to the travelling salesman problem. *Nature*, 326:689–691.

Field, D. J. (1987). Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America*, 4:2379–2394.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172.

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

Fritzke, B. (1991). Unsupervised clustering with growing cell structures. In *Proceedings of IJCNN-91*, volume 2, pages 531–536, Seattle.

Galland, C. (1992). *Self-organization using Deterministic Boltzmann Machine learning*. PhD thesis, Physics Department, University of Toronto.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.

Georgopoulos, A. P., Schwartz, A. B., and Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, 243:1416–1419.

Gray, R. M. (1984). Vector quantization. *IEEE ASSP Magazine*, 4:4–28.

Grenander, U. (1982). *Lectures in Pattern theory I, II and III: Pattern Analysis, Pattern synthesis and regular structures, 1976-81*. Springer-Verlag.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121–134.

Gull, S. F. (1989). Developments in maximum entropy data analysis. In Skilling, J., editor, *Maximum Entropy and Bayesian Methods, Cambridge, 1988*, pages 53–71. Kluwer, Dordrecht.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the theory of neural computation.* Addison-Wesley Publishing Company, Redwood City, CA.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst, MA.

Hinton, G. E. (1989a). Connectionist learning procedures. *Artificial Intelligence*, 40(1):185–234.

Hinton, G. E. (1989b). Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation*, 1(1):143–150.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E., McClelland, J. L., and the PDP research group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 77–109. MIT Press, Cambridge, MA.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann Machines. In Rumelhart, D. E., McClelland, J. L., and the PDP research group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 282–317. MIT Press, Cambridge, MA.

Hinton, G. E. and van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *Sixth ACM Conference on Computational Learning Theory*, Santa Cruz, CA.

Intrator, N. (1993). Combining exploratory projection pursuit and projection pursuit regression with application to neural networks. *Neural Computation*, 5(3):443–455.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Jaynes, E. T. (1986). Bayesian methods: General background. In Justice, J., editor, *Maximum Entropy and Bayesian Methods in Applied Statistics*, pages 1–25. Cambridge University Press, Cambridge.

Jordan, M. and Rumelhart, D. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354.

Kendall, G. and Hall, T. (1993). Optimal network construction by minimum description length. *Neural Computation*, 5(2):210–212.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69.

Kohonen, T. (1989). *Self-Organization and Associative Memory, 3rd edition.* Springer-Verlag, Berlin.

Kohonen, T., Mäkisara, K., and Saramäki, T. (1984). Phonotopic maps—Insightful representation of phonological features for speech recognition. *Proceedings of the Seventh International Conference on Pattern Recognition*, pages 182–185.

Konishi, M., Takahashi, T. T., Wagner, H., Sullivan, W., and Carr, C. (1988). Neurophysiological and anatomical substrates of sound localization in the owl. In *Auditory Function*, pages 721–745. G.M. Edelman and W.E. Gall and W.M. Cowan, New York.

Kramer, M. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37:233–243.

Kullback, S. (1959). *Information Theory and Statistics*. Wiley, New York.

Leclerc, Y. G. (1989). Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3(1):73–102.

Lee, C., Rohrer, W. R., and Sparks, D. L. (1988). Population coding of saccadic eye movements by neurons in the superior colliculus. *Nature*, 332:357–360.

Linde, Y., Buzo, A., and Gray, R. (1980). An algorithm for Vector Quantizer design. *IEEE Transactions on Communications*, COM-28:84–95.

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

Luttrell, S. P. (1990). Derivation of a class of training algorithms. *IEEE Transactions on Neural Networks*, 1:1229–232.

MacKay, D. (1992a). Bayesian interpolation. *Neural Computation*, 4(3):415–447.

MacKay, D. (1992b). A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

Martinetz, T. and Schulten, K. (1991). A 'neural gas' network learns topologies. In *Proceedings of ICANN-91*, pages 397–402.

Michalski, R. S. and Stepp, R. (1983). Learning from observation: Conceptual clustering. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An artifical intelligence approach*. Tioga, Palo Alto, CA.

Mitchell, T., Keeler, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80.

Mozer, M. C. (1991). Discovering discrete distributed representations with iterative competitive learning. In *Advances in Neural Information Processing Systems 3*, pages 627–634, San Mateo, CA. Morgan Kaufmann.

Mumford, D. (1992). Pattern theory: A unifying perspective. In *Proceedings of the First European Congress of Mathematics*, Paris.

Neal, R. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

Nowlan, S. J. (1988). Auto-encoding with entropy constraints. In *Proceedings of INNS First Annual Meeting*, pages 369–376, Boston, MA.

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

Nowlan, S. J. and Hinton, G. E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493.

Obermayer, S. J., Ritter, H., and Schulten, K. (1991). A neural network model for the formation of the spatial structure of retinotopic maps, orientation and ocular dominance columns. In Kohonen, T., Simula, O., and Kangas, J., editors, *Artificial Neural Networks I*, pages 505–511. North Holland.

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.

Oja, E. (1991). Data compression, feature extraction, and autoassociation in feedforward neural networks. In Kohonen, T., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pages 737–745. North Holland.

Parker, D. B. (1985). Learning-logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pentland, A. (1989). Part segmentation for object recognition. *Neural Computation*, 1(1):82–91.

Peterson, C. and Anderson, J. R. (1987). A mean field theory learning algorithm for neural nets. *Complex Systems*, 1:995–1019.

Quinlan, J. and Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248.

Rissanen, J. (1978). A predictive least squares principle. Technical Report IBM Res., K54, IBM.

Rissanen, J. (1985). Minimum description length principle. In *Encyclopedia of Statistical Sciences*, volume 5, pages 523–527. Wiley and Sons, New York.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Co., Singapore.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., and the PDP research group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 318–362, Cambridge, MA. MIT Press.

Rumelhart, D. E. and Zipser, D. (1985). Feature discovery by competitive learning. *Cognitive Science*, 9:75–112.

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

Saund, E. (1989). Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314.

Schmidhuber, J. H. (1992). Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879.

Segen, J. (1990). Graph clustering and model learning by data compression. In *Proceedings of the 7th International Conference on Machine Learning*, pages 93–101, Austin, Texas.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656.

Suga, N. (1984). The extent to which biosonar information is represented in the bat auditory cortex. In *Dynamic aspects of neocortical function*, pages 315–373, New York. Wiley.

Usui, S., Nakauchi, S., and Nakano, M. (1991). Internal color representation acquired by a five-layer neural network. In Kohonen, T., Simula, O., and Kangas, J., editors, *Artificial Neural Networks I*, pages 867–872. North Holland.

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

Watanabe, S. (1985). *Pattern Recognition: Human and Mechanical.* Wiley, New York.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In Lippman, R. P., Moody, J., and Touretzky, D., editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. Morgan Kaufmann, San Mateo, CA.

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

Willshaw, D. and von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London B*, 194:431–445.

Zipser, D. (1986). Programming networks to compute spatial functions. Technical Report ICS Report 8606, UCSD.