To appear in: *Neural Computation, 7:3*, 1995.

Learning Population Codes by Minimizing Description Length

Richard S. Zemel
Computational Neurobiology Laboratory
The Salk Institute
10010 North Torrey Pines Rd.
La Jolla, CA 92037

Geoffrey E. Hinton
Department of Computer Science
University of Toronto
6 King's College Road
Toronto, Ontario M5S 1A4

Abstract

The Minimum Description Length principle (MDL) can be used to train the hidden units of a neural network to extract a representation that is cheap to describe but nonetheless allows the input to be reconstructed accurately. We show how MDL can be used to develop highly redundant population codes. Each hidden unit has a location in a low-dimensional *implicit* space. If the hidden unit activities form a bump of a standard shape in this space, they can be cheaply encoded by the center of this bump. So the weights from the input units to the hidden units in an autoencoder are trained to make the activities form a standard bump. The coordinates of the hidden units in the implicit space are also learned, thus allowing flexibility, as the network develops a discontinuous topography when presented with different input classes.

1 Introduction

Most existing unsupervised learning algorithms can be understood using the Minimum Description Length (MDL) principle (Rissanen, 1989). Given an ensemble of input vectors, the aim of the learning algorithm is to find a method of coding each input vector that minimizes the total cost, in bits, of communicating the input vectors to a receiver. There are three terms in the total description length:

- The **code-cost** is the number of bits required to communicate the code that the algorithm assigns to each input vector.
- The **model-cost** is the number of bits required to specify how to reconstruct input vectors from codes (e.g., the hidden-to-output weights).
- The **reconstruction-error** is the number of bits required to fix up any errors that occur when the input vector is reconstructed from its code.

Formulating the problem in terms of a communication model allows us to derive an objective function for a network (note that we are not actually sending the bits). For example, in competitive learning (vector quantization), the code is the identity of the winning hidden unit, so by limiting the system to $\mathcal{H}$ units we limit the average code-cost to at most $\log_2 \mathcal{H}$ bits. The reconstruction-error is proportional to the squared difference between the input vector and the weight-vector of the winner, and this is what competitive learning algorithms minimize. The model-cost is usually ignored.

The representations produced by vector quantization contain very little information about the input (at most $\log_2 \mathcal{H}$ bits on average). To get richer representations we must allow many hidden units to be active at once and to have varying activity levels. Principal components analysis (PCA) achieves this for linear mappings from inputs to codes. It can be viewed as a version of MDL in which we limit the code-cost by only having a few hidden units, and ignore the model-cost and the accuracy with which the hidden activities must be coded. An autoencoder that tries to reconstruct the input vector on its output units will perform a version of PCA if the output units are linear. We can obtain novel and interesting unsupervised learning algorithms using this MDL approach by considering various alternative methods of communicating the hidden activities. The algorithms can all be implemented by backpropagating the derivative of the code-cost for the hidden units in addition to the derivative of the reconstruction-error backpropagated from the output units.

Any method that communicates each hidden activity separately and independently will tend to lead to *factorial* codes because any mutual information between hidden units will cause redundancy in the communicated message, so the pressure to keep the message short will squeeze out the redundancy. In (Zemel, 1993) and (Hinton and Zemel, 1994), we present algorithms derived from this MDL approach aimed at developing factorial codes. Although factorial codes are interesting, they are not robust against hardware failure nor do they resemble the population codes found in some parts of the brain. Our aim in this paper is to show how the MDL approach can be used to develop population codes in which the activities of hidden units are highly correlated.

2 Population Codes

2.1 Constraint surfaces

Unsupervised algorithms contain an implicit assumption about the nature of the structure or constraints underlying the input set. For example, competitive learning algorithms are suited to datasets in which each input can be attributed to one of a set of possible causes. In the algorithm we present here, we assume that each input can be described as a point in a low-dimensional continuous *constraint space*.

An example is shown in Figure 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 a particular image amongst a set of images of the hippo from multiple viewpoints can be concisely represented by first describing a canonical hippo, and then encoding the instance as a point in the constraint space spanned by the four 2D viewing parameters: (x, y)-position, orientation, and scale. Other examples exist in biology, e.g., 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). Averaging each cell's preferred motion

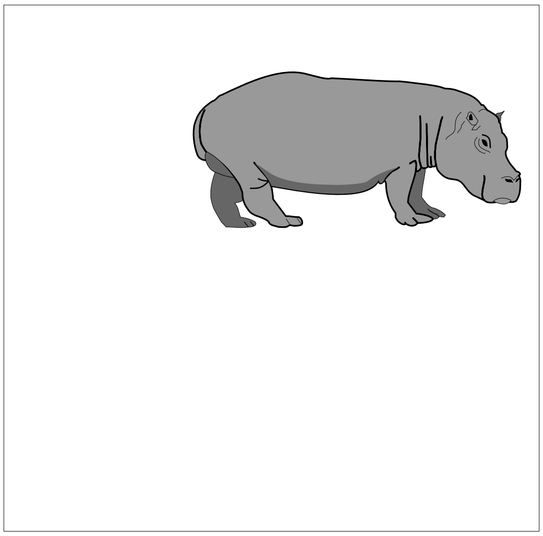 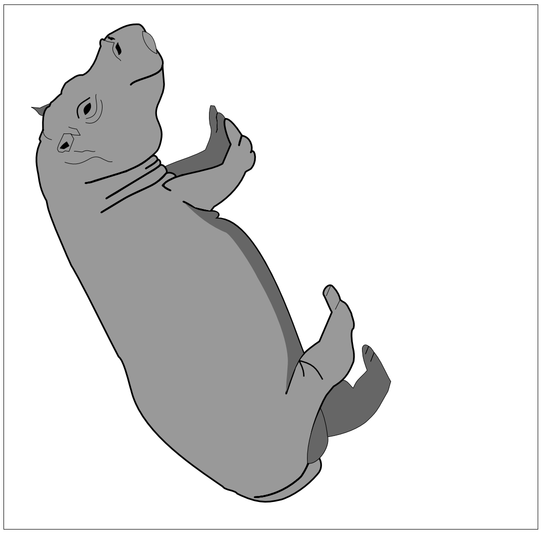

Figure 1: Two example images where only a few dimensions of variability underlie seemingly unrelated vectors of pixel intensity values.

directions weighted by its activity accurately predicts the direction of movement that the animal makes. In these examples, knowledge of an underlying lower-dimensional constraint surface allows for compact descriptions of stimuli or responses. Our goal is to find and represent the constraint space underlying high-dimensional data samples.

2.2 Representing underlying dimensions using population codes

In order to represent inputs as points drawn from a constraint space, we choose a *population code* style of representation. In a population code, each code unit is associated with a position in what we call the *implicit space*, and the code units' pattern of activity conveys a single point in this space. This implicit space should correspond to the constraint space. For example, suppose that each code unit is assigned a position in a 2D implicit space, where one dimension corresponds to the size of the shape and the second to its orientation (see Figure 2). A population of code units broadly-tuned to different positions can represent any particular instance of the shape by their relative activity levels.

This example illustrates that population codes involve *three* quite different spaces:

1. the input-vector space (the pixel intensities in the example);
2. the hidden-vector space (where each hidden, or code unit entails an additional dimension);
3. the implicit space, which is of lower dimension than the other two spaces.

In a learning algorithm for population codes, the implicit space is intended to come to smoothly represent the underlying dimensions of variability in the inputs, i.e., the constraint space.

For instance, in a Kohonen network (Kohonen, 1982), the hidden unit with the greatest total input has an activity of one, while the others are zero. Yet these hidden units are also assigned positions in implicit space based on a neighborhood function that determines the degree of interaction

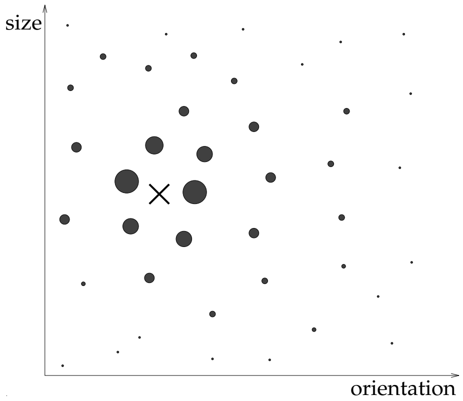

Figure 2: The population code for an instance in a two-dimensional implicit space. The position of each blob corresponds to the position of a unit within the population, and the blob size corresponds to the unit's activity. Here one dimension describes the size and the other the orientation of a shape. We can determine the instantiation parameters of this particular shape by computing the center of gravity of the blob activities, marked here by an "X".

between a pair of units according to their distance. The active unit then maps the input to a particular point in this implicit space. Here the implicit space topology is defined *a priori* through fixed neighborhood relations, and the algorithm then adjust weights so that neighbors in implicit space respond to similar inputs. Similarly, in a 1-dimensional version of the elastic-net algorithm (Durbin and Willshaw, 1987), the code 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 code 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.

Population codes have several computational advantages, in addition to their obvious biological relevance. The codes contain some redundancy and hence have some degree of fault-tolerance. A population code as described above reflects structure in the input, in that similar inputs are mapped to nearby implicit positions, if the implicit dimensionality matches the intrinsic dimensionality of the input. They also possess a hyperacuity property, as the number of implicit positions that can be represented far exceeds the number of code units; this makes population codes well-suited to describing values along a continuous dimension.

3 Learning Population Codes With MDL

Autoencoders are a general way of addressing issues of coding. The hidden unit activities for an input are the codes for that input which are produced by the input-hidden weights, and reconstruction from the code is done by the hidden-output mapping. In order to allow an autoencoder to develop population codes for an input set, we need some additional structure in the hidden layer that will allow a code vector to be interpreted as a point in implicit space. While most topographic-map formation algorithms (e.g., the Kohonen and elastic-net algorithms)

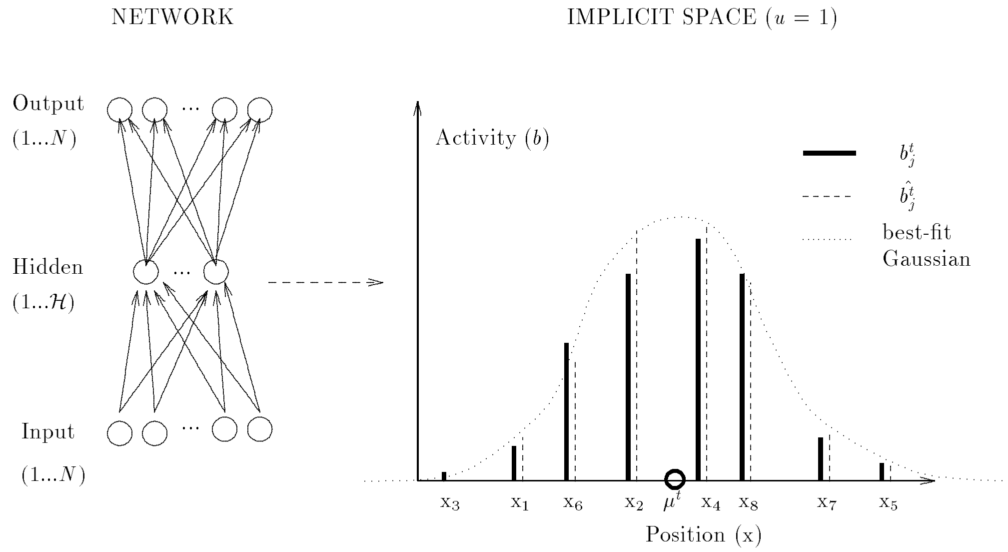

Figure 3: Each of the $\mathcal{H}$ hidden units in the autoencoder has an associated position in implicit space. Here we show a 1D implicit space. The activity b_j^t of each hidden unit j on case t is shown by a solid line. The network fits the best Gaussian to this pattern of activity in implicit space. The predicted activity, $\hat{b}_j^t$, of unit j under this Gaussian is based on the distance from x_j to the mean μ^t; it serves as a target for b_j^t.

define the topology of this implicit space by fixed neighborhood relations, in our algorithm we use a more explicit representation. Each hidden 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. To determine what implicit position is represented by a vector of hidden activities, we can average together the implicit coordinates of the hidden units, weighting each coordinate vector by the activity level of the unit.

Suppose, for example, that each hidden unit is connected to an 8x8 retina and has 2 implicit coordinates that represent the size and orientation of a particular kind of shape on the retina, as in our earlier example. If we plot the hidden 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 3 depicts this for a 1D implicit space).

If the activities form a perfect Gaussian bump of fixed variance 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 hidden units.

It is important to realize that the activity of a hidden 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.

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 vector of hidden activities and weights.

3.1 Activations and objective function

Let a_i^t be the activity of input unit i on case t. The actual activity of a hidden unit j is then:

$$b_j^t = \exp(\sum_i w_{ji} a_i^t) / \sum_{h=1}^{H} \exp(\sum_i w_{hi} a_i^t) \tag{1}$$

Note that the unit's actual activity is independent of its position x_j in implicit space. Its expected activity is its normalized value under the predicted Gaussian bump:

$$\hat{b}_j^t = \exp(-(x_j - \mu^t)^2 / 2\sigma^2) / \sum_{i=1}^{\mathcal{H}} \exp(-(x_i - \mu^t)^2 / 2\sigma^2) \tag{2}$$

where μ^t is the mean of the bump and σ its width, which we assume is fixed throughout training. The activity c_k^t of output unit k is just the weighted sum of its inputs. The network has full inter-layer connectivity.

Currently, we ignore the model-cost, and assume a fixed cost for specifying the bump mean on each case, so the description length to be minimized is:

$$\begin{aligned} E^t &= B^t + R^t \\ &= \sum_{j=1}^{\mathcal{H}} (b_j^t - \hat{b}_j^t)^2 / 2V_B + \sum_{k=1}^{N} (a_k^t - c_k^t)^2 / 2V_R \end{aligned} \tag{3}$$

where V_B and V_R are the fixed variances of the Gaussians used for coding the bump-errors[1] and the reconstruction-errors.

We have explored several methods for computing μ^t, the mean of the bump. Simply computing the center of gravity of the hidden 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 μ^t; it is the position in implicit space that minimizes B^t given $\{x_j, b_j^t\}$.

Both the network weights and the implicit coordinates of the hidden units are adapted simultaneously. We minimize E with respect to the weights by back-propagating the derivatives of the reconstruction-error from the output units, and then add in the derivatives of the bump-error at the hidden layer. The implicit coordinates affect the bump-error through the computation of the predicted hidden activities (see Equation 2).

[1] Note that this Gaussian assumption for encoding the bump-error is an approximation to the true distribution. 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.

4 Experimental Results

4.1 Parameters

The algorithm includes three adjustable parameters. The width of the Gaussian bump in implicit space, σ, 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, the elastic-net, and the mixture-of-Gaussians, this width is initially large and is then annealed down to a minimum width during training; this approach did not significantly improve this algorithm's performance. The second and third parameters are V_B and V_R, the variances of the Gaussians for coding 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.

Two architectural parameters also play important roles in this algorithm. The first is the number of hidden units. In order to accurately and robustly represent a wide range of values in implicit space, the network must contain a sufficient number of hidden 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 hidden units, and the input is devoid of noise, then the algorithm should 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.

4.2 Experiment 1: Learning to represent position

Each 8x8 real-valued input image contains a rigid object, which is composed of two Gaussian blobs of intensity at its ends (see Figure 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 parameters describe the variability due to seeing the same rigid 2D object from different viewpoints in the fronto-parallel plane. In order to avoid edge effects, the input space was represented using wraparound. We also use wraparound in the implicit space, which creates a toroidal shape, i.e., the points at 2π radians are neighbors of the points at 0 radians.

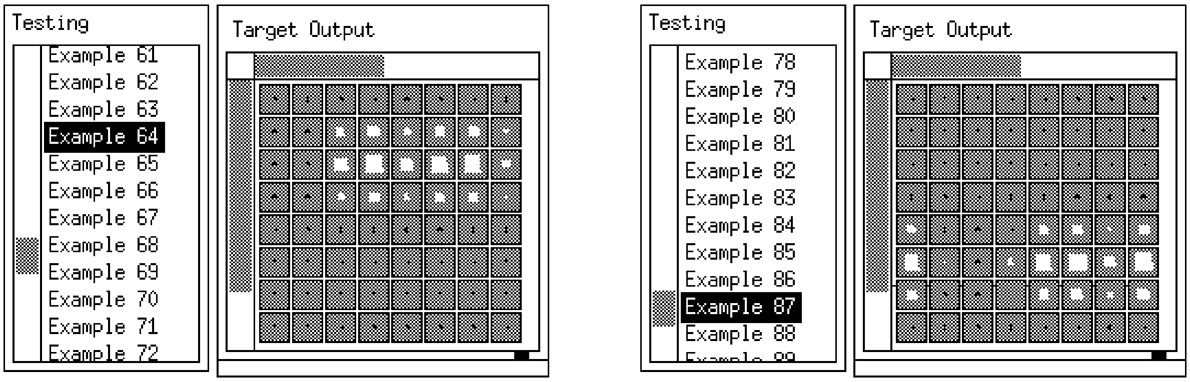

Figure 4: Each 8x8 real-valued input image for the first experiment contains an instance of a dipole. The dipole has four continuous degrees-of-freedom: (x, y)-position, orientation, and size. This figure shows two sample images from the test set. The image on the right shows that the input space is represented using wraparound.

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 hidden units has random 2D implicit coordinates. The system converges to a minimum of the objective function after 25 epochs. The *generalization length* (the mean description length 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.

Each hidden unit develops a *receptive field* so that it responds to objects in a limited neighborhood of the underlying constraint space that corresponds to its learned position in implicit space (see Figure 5). This arises due to the constraint that the implicit pattern of activity should be bump-like. The algorithm successfully learns the underlying constraint surface in this dataset; implicit space forms a relatively smooth, stable map of the generating surface (Figure 6), 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.

4.3 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.

We give each hidden unit three implicit coordinates, and also increase the number of hidden units to 225. Since this network has a larger ratio of hidden to input units, we increase the bump variance ($V_B = 2.5$) in order to maintain a balance between the two costs. The network converges after 60

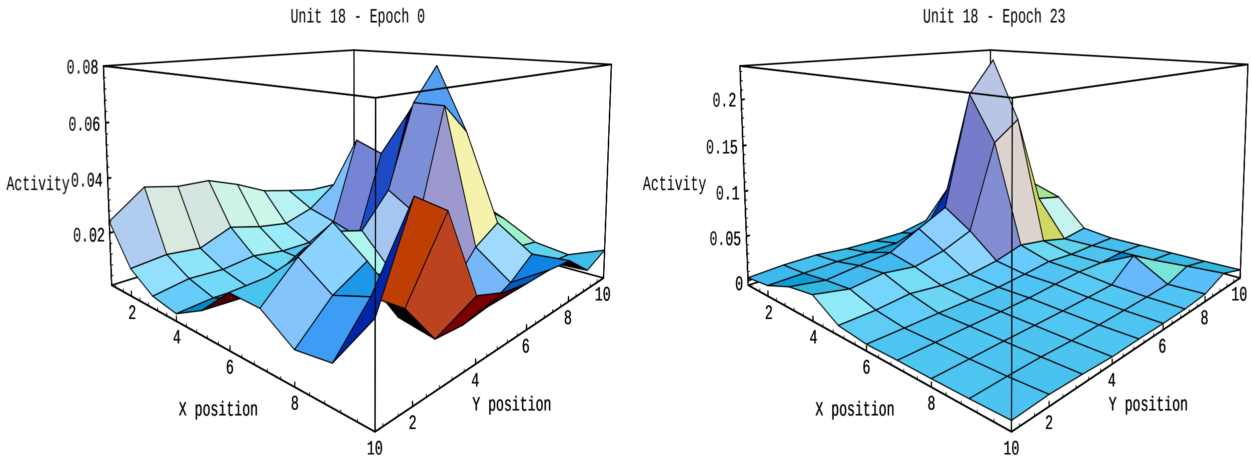

Figure 5: This figure shows the receptive field in implicit space for two hidden units. Here the 2 dimensions in implicit space correspond to x and y positions. The left panel shows that before learning, the units respond randomly to 100 different test patterns, generated by positioning a shape in the image at each point in a 10x10 grid. The right 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.

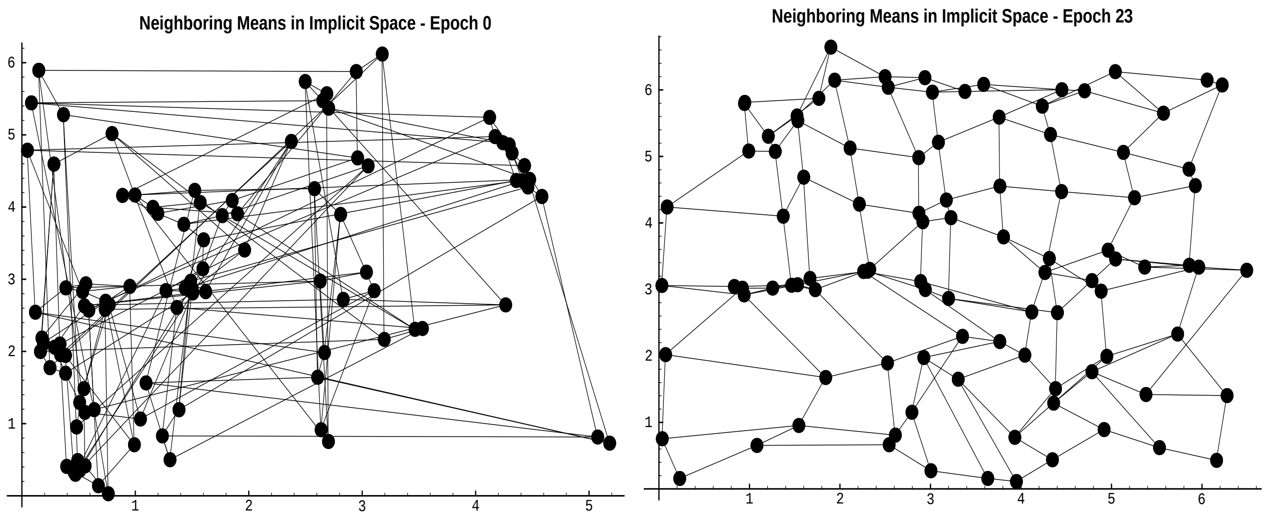

Figure 6: This figure shows the implicit positions of the bump means for the test set before and after training. In the 100 testing examples, the object is located at the gridpoints of a 10x10 grid covering the space of possible positions. In this figure, lines connect the means between a given test image and its 4 neighbors on this grid. Note that implicit space has a toroidal shape, i.e., the points at 2π 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.

epochs of training on the 1000 images. The generalization length is 1.16 bits per image. The hidden unit activities again form a population code that allow the input to be accurately reconstructed. 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. While the algorithm often gets stuck in local minima when we decrease the number of hidden units below 200, this problem virtually disappears with sufficient hidden units. The algorithms defined using this MDL approach can effectively remove the excess units once good representations for the input set have been discovered with the initial large pool of units.

4.4 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 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 hidden 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 hidden units trained on a single shape in various positions. The decrease in representational quality can be attributed to this additional "what" dimension of the training set.

After training, we find that one set of hidden 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 7). The network 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. A wide variety of this solution type can be found, based on parameters such as the within-shape versus between-shape correlations. For the training set used in this experiment, this second solution typically has a higher generalization length (≈ 1.8 bits).

A Kohonen network 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. The network always forms a patchy topographic map, where the underlying constraint surface is captured in a local fashion by the map. While initializing the Kohonen algorithm with a large neighborhood size and using a long annealing schedule may enable it to learn the other representation, the key point is that learning this sort of mapping is difficult, which is probably due 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 hidden units *learn* their implicit coordinates in our algorithm allows 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

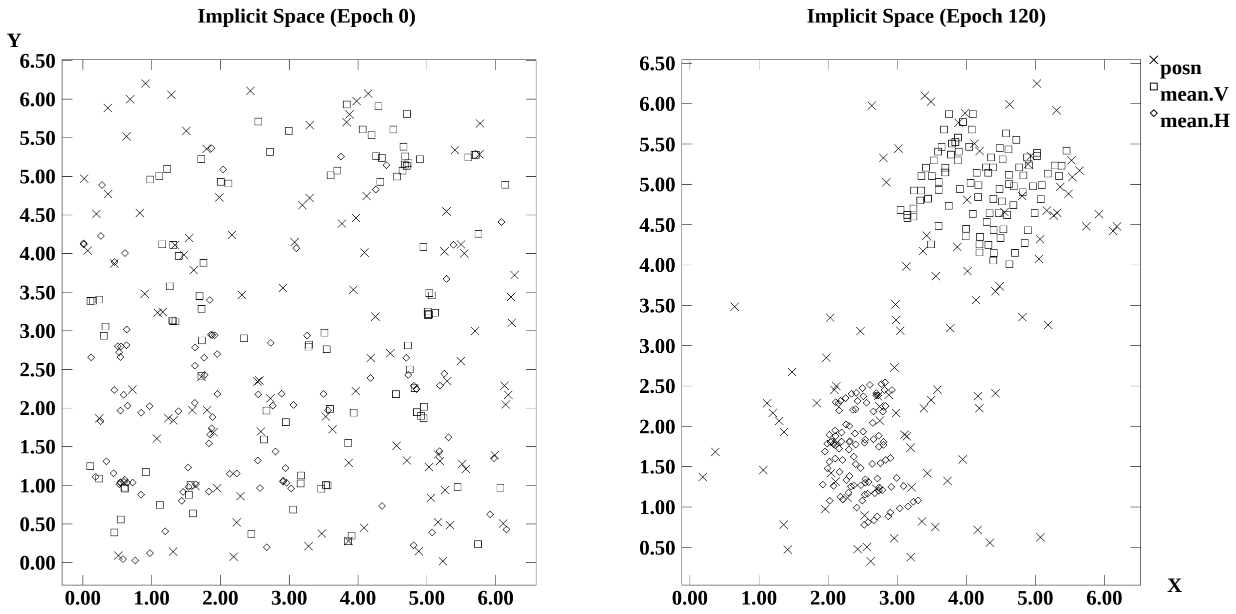

Figure 7: This figure shows the positions of the hidden units and the means in the 2D implicit space before and after training on the horizontal/vertical task. In each plot, an "x" marks the implicit position of a hidden unit; a box marks the implicit mean of the bump formed for an input image containing 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 hidden 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.

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 found by the Kohonen network. Obermayer, Ritter, and Schulten (1991) showed how a Kohonen algorithm trained on this task could develop a similar map. Goodhill and Willshaw (1990) discussed the solutions that may be found using an elastic net algorithm, while Dayan (1993) analyzed how the neighborhood relations in the elastic net could determine the learned topology. Each of these papers analyzes the dependence of the map structure on certain key underlying input parameters; these analyses apply to the task described here as well.

5 Related work

This new algorithm bears some similarities to several earlier algorithms, particularly topographic map formation algorithms such as the Kohonen and elastic-net algorithms. Like these algorithms, our method aims to create a map where the constructed, or implicit, space corresponds to the underlying constraint space of the inputs; this structure is created primarily by indirect local learning.

Several important differences exist between our method and these earlier algorithms. The key difference is that our algorithm explicitly encourages the hidden unit activity to form a population code in implicit space rather than developing these codes implicitly through neighborhood interaction during learning.

In addition, the population code in these methods is in effect formed in input space. Each unit's weights are moved towards the input patterns, whereas our method moves the weights based on the implicit space coding as well as the reconstruction error.[2]

In (Saund, 1989), hidden unit patterns of activity in an autoencoder are trained to form Gaussian bumps, where the center of the bump is intended to correspond to the position in an underlying dimension of the inputs. Ossen (1992) proposed a similar objective, and our activation function for the hidden layer also resembles the one he used. Yet the objective function in our algorithm is quite different due to the implicit space construction. An additional crucial difference exists: in these earlier algorithms[3], the implicit space topology is statically determined *a priori* by the ordering of the hidden units, while units in our model learn their implicit coordinates. Learning this topology lends additional flexibility to our algorithm.

[2] 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.

[3] 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. Bregler and Omohundro (1994) present a method of learning the dimensionality using local linear patches.

6 Discussion and current directions

We have shown how MDL can be used to develop non-factorial, redundant representations. The objective function is derived from a communication model where rather than communicating each hidden unit activity independently, we instead communicate the location of a Gaussian bump in a low-dimensional implicit space. If the hidden 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 are the instantiation parameters of an object, this implicit space comes to correspond to these parameters. Since the implicit coordinates of the hidden units are also learned, the network develops separate population codes when presented with different objects.

While we have tested the algorithm on noisier versions of the datasets described above, and have found that the solution quality gracefully degrades with added noise, we have not described any results of applying it to more realistic data. Instead we have chosen to emphasize this hand-crafted data in order to determine the quality of network solution. The primary contributions of this paper are theoretical: we introduce a method of encouraging a particular functional form of activity for the hidden units, and also demonstrate an objective based on compact coding that nevertheless encourages redundancy in the codes. It would be interesting to consider generalizations of this algorithm which derive from positing other functional forms for the hidden unit activity patterns.

Our method can 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. Alternatively, instead of using multiple hidden layers to extract a single code for the input, one could use a hierarchical system in which the code-cost is computed at every layer.

A limitation of this approach (as well as the aforementioned approaches) is the need to predefine the dimensionality of implicit space. We are currently working on an extension that will allow the learning algorithm to determine for itself the appropriate number of dimensions in implicit space. We start with many dimensions but include the cost of specifying μ^t in the description length. This depends on how many implicit coordinates are used. If all of the hidden units have the same value for one of the implicit coordinates, it costs nothing to communicate that value for each bump. In general, the cost of an implicit coordinate depends on the ratio between its variance (over all the different bumps) and the accuracy with which it must be communicated. So the network can save bits by reducing the variance for unneeded coordinates. This creates a smooth search space for determining how many implicit coordinates are needed.

7 Acknowledgements

We thank Peter Dayan, Klaus Obermayer, and Terry Sejnowski for their help. This research was supported by grants from NSERC, the Ontario Information Technology Research Center, and the Institute for Robotics and Intelligent Systems. Geoffrey Hinton is the Noranda Fellow of the Canadian Institute for Advanced Research.

References

Bregler, C. and Omohundro, S. M. (1994). Surface learning with applications to lipreading. In *Advances in Neural Information Processing Systems 6*, pages 43–50, San Mateo, CA. Morgan Kaufmann.

Dayan, P. (1993). Arbitrary elastic topologies and ocular dominance. *Neural Computation*, 5(3):392–401.

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

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

Goodhill, G. J. and Willshaw, D. J. (1990). Application of the elastic net algorithm to the formation of ocular dominance stripes. *Network*, 1:41–61.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and Helmholtz free energy. In *Advances in Neural Information Processing Systems 6*, pages 3–10, San Mateo, CA. Morgan Kaufmann.

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

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

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

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.

Ossen, A. (1992). Learning topology-preserving maps using self-supervised backpropagation on a parallel machine. Technical Report TR-92-059, International Computer Science Institute.

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

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

Zemel, R. S. (1993). *A Minimum Description Length Framework for Unsupervised Learning*. PhD thesis, University of Toronto.