Abstraction and Representation of Continuous Variables in Connectionist Networks

Eric Saund
Department of Brain and Cognitive Sciences and
the Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

A method is presented for using connectionist networks of simple computing elements to discover a particular type of constraint in multidimensional data. Suppose that some data source provides samples consisting of n-dimensional feature-vectors, but that this data all happens to lie on an m-dimensional surface embedded in the n-dimensional feature space. Then occurrences of data can be more concisely described by specifying an m-dimensional location on the embedded surface than by reciting all n components of the feature vector. The recoding of data in such a way is a form of abstraction. This paper describes a method for performing this type of abstraction in connectionist networks of simple computing elements. We present a scheme for representing the values of continuous (scalar) variables in subsets of units. The backpropagation weight updating method for training connectionist networks is extended by the use of auxiliary pressure in order to coax hidden units into the prescribed representation for scalar-valued variables.

1 Introduction

A key theme in Artificial Intelligence is to discover good representations for the problem at hand. A good representation makes explicit information useful to the computation, it strips away obscuring clutter, it reduces information to its essentials.

An important step in generating a good representation is to expose constraint in the information to be dealt with. Knowledge of constraint allows one to condense a great deal of unwieldy data into a concise description. In the structure from motion problem in vision, for example [Ullman, 1979], the movement of a large number of dots in an image can be described concisely in terms of the motion of a single coordinate frame—provided one possesses knowledge of the constraint, namely, the rigid relative locations of the points in space. For many information processing tasks such as storage, transmission, and matching to memory, a concise description of information is preferable to a redundant one.

Often, constraint in a particular domain is found by careful analysis of the problem. One may discover causal relationships between variables, or, as in the structure from motion case, mathematical formulations that capture the structure of the problem.

But for many problems there may exist no elegant or transparent expression of the constraint operating. Figure 1 illustrates a sample of (x, y) points generated by some unknown data source. Evidently, the source operates under some constraint because all data points appear to lie on a one-dimensional curve. If one possesses knowledge of this curve then one may express data samples with only one number, the location along the curve, instead of the two numbers required to spell out the (x, y) coordinates. However, location along the curve in figure 1 cannot be calculated from x and y by any formula because the curve has no simple analytical description. In order to take advantage of constraint implicit in the data, we must have some way of using knowledge of the underlying curve. This is the problem addressed by this paper: to provide the means to capture constraint in multidimensional data, even when the constraint is of such arbitrary form as that in figure 1.

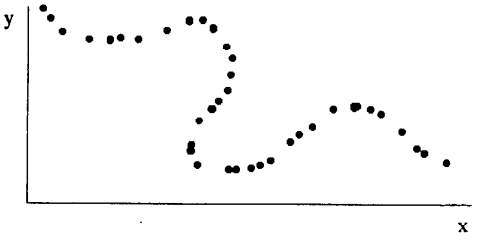


Figure 1. Samples of two-dimensional data constrained to lie on a one-dimensional curve.

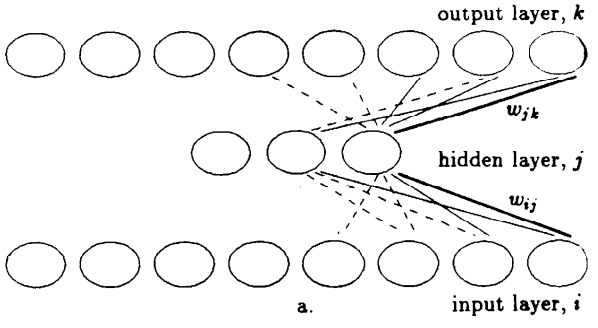
This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124. The author is supported by a fellowship from the NASA Graduate Student Reseachers Program.

2 Background

This form of data abstraction has been called "dimensionality reduction" by Kohonen [1984]. The problem is to define a coordinate system on an m-dimensional surface embedded in an n-dimensional space, plus a mapping between this coordinate system and that of the embedding space, when n-dimensional data samples are all drawn from locations on the embedded surface. Kohonen has implemented a method for performing dimensionality reduction which is modeled after a theory of the self-organization of topographic mappings by cells in the brain. Kohonen's method suffers from a number of drawbacks. The result delivered by this method confounds the shape of any underlying, lower-dimensional, constraint surface with the probability distribution of data samples over that surface. Furthermore, the computation of any given data sample's description in terms of location on the constraint surface requires explicit search over all locations on that surface.

We propose a different mechanism for achieving dimensionality reduction using connectionist networks of simple computing elements. Of particular interest are demonstrations by Hinton, et al [1984], and Rumelhart, et al [1985], that "hidden units" in these networks are able to attain abstract representations capturing constraint in binary input data.

A prototypical example is the encoder problem (see figure 2). Here, the activity, a_k, in a layer of eight output units is calculated from the activity, a_j in the middle layer



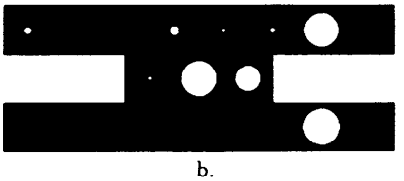


Figure 2. a. 8-3-8 network. Activity at the input layer drives hidden layer. Activity in the hidden layer drives output. b. Activity in an 8-3-8 network trained for the encoder problem. Input is constrained so that only one input unit is ON at a time. Activity at output matches input. The information as to which input unit is ON is able to be transmitted via the hidden unit layer of only three units. Size of circle represents unit's activity.

of three units:

$$a_k = f(s_k) = f(\sum_j w_{jk} a_j), \tag{1}$$

where w_{jk} is the connection weight between the jth middle layer unit and the kth output layer unit. The activity in the middle layer is calculated from the activity in the input layer in a similar way. The middle layer units are called "hidden" units because their activity is not directly accessible either at network input or output. f is typically a sigmoidal nonlinear function, for example,

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The goal of the encoder problem is to set the weights such that if any single unit in the input layer is set ON (near 1), and the rest set OFF (near 0), the activity will propagate so that only the corresponding output unit will be ON. Because data presented to the input layer is constrained, so that only a subset of all possible patterns of activity in the input layer ever actually appear, the information as to which unit is ON may be propagated to the output layer through a middle layer containing *fewer* than eight units. In particular, the three middle layer units may transmit the information by adopting a binary code. Rumelhart, Hinton, and their colleagues have described a method, called *backpropagation*, for training a network to achieve such behavior without directly specifying the behavior of the hidden layer: repeatedly present the network with sample input and allow activity to propagate to the output layer, observe the error between the desired output and the observed activity at the output layer, and use these errors to update the network weights. This method is described in more detail below.

This paper describes a method for extending the backpropagation weight updating scheme for connectionist networks in order to perform dimensionality reduction over continuous-valued data.

3 Representing Scalar Variables

We must start by endowing networks with the ability to represent not just the binary variables, ON and OFF, but variables continuous over some interval. For convenience let this interval be (0,1).

One conceivable way of representing scalars in simple units is via a unit's activity level itself. Since only one weight connects any middle-layer unit to any given output unit, this strategy is clearly inadequate for representing anything but linear relationships between variables. The relationship between x and y in figure 1 is not linear, so the relationship between x and some hidden variable, u, and between y and u must not both be linear.

Another possibility is to quantize the scalar variables and let individual units represent successive intervals. Be-

cause quantized numbers can only be as accurate as the number of units (and therefore intervals) provided, we suggest modifying this tactic somewhat by allowing different units to represent different subintervals, but then smearing activity over units covering nearby subintervals. Figure 3a shows such a smearing function, S_ω, which in this case happens to be the derivative of the function, f, of equation (2). Other smearing functions, such as the gaussian, may be used. The parameter, ω, controls the width of the smearing effect.

The number expressed in a pattern of activity may be decoded as the placement of the smearing function, S, at the location, x, within the interval, which minimizes the least-square difference (figure 3b),

$$Q(x) = \sum_i (S_{x-i} - a_i)^2 \tag{3}$$

Under this method of encoding continuous numbers, resolution of better than 1 part in 50 may be achieved using eight units. For the remainder of this paper we refer to any subset of units whose activity displays this characteristic form for encoding scalar values as exhibiting "scalarized" behavior.

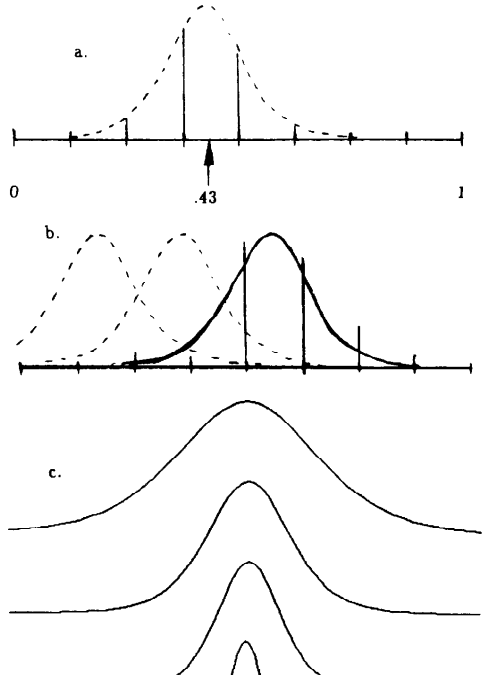


Figure 3. a. Activity pattern in a set of 9 units representing the number, .43. Profile is of the smearing function, S_ω. b. The number represented by the activity in a set of units is found by sliding the profile of the smearing function, S, along the number line to the location where it best fits the activity in the units. c. Smearing function, S_ω, for several values of the smearing width parameter, ω.

4 Training the Network

The backpropagation method for training connected networks to exhibit some desired input/output behavior may be derived by expressing the relationship between a.) a cost for error in output behavior, and b.) the strengths of individual weights in the network. Following Rumelhart, et al [1985], define cost

$$E = \sum_p E_p = \sum_p \sum_k (a_{pk} - t_{pk})^2 = \sum_p \sum_k \delta_{pk}^2 \tag{4}$$

as the cost over all output units, k, of error between output a_k and target output, t_k, summed over all sets of presentations, p, of input data. Weights will be adjusted a slight amount at each presentation, p, so as to attempt to reduce E_p. The amount to adjust each weight connecting a middle layer unit and an output unit is proportional to (from (1) and (4))

$$\frac{\partial E_p}{\partial w_{jk}} = \frac{\partial E_p}{\partial a_k}\frac{\partial a_k}{\partial s_k}\frac{\partial s_k}{\partial w_{jk}} = \delta_{pk} f'(s_k) a_j \tag{5}$$

Take

$$\Delta w_{jk} = \eta \delta_{pk} f'(s_k) a_j \tag{6}$$

as the amount to adjust weight w_{jk} at presentation p. η is a parameter controlling learning rate.

Adjustments of weights between the input and middle layers is proportional to

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial s_j}\frac{\partial s_j}{\partial w_{ij}} \tag{7}$$

$$= \left(\sum_k \frac{\partial E_p}{\partial a_k}\frac{\partial a_k}{\partial s_k}\frac{\partial s_k}{\partial w_{jk}}\right)\frac{\partial a_j}{\partial s_j}\frac{\partial s_j}{\partial w_{ij}} \tag{8}$$

$$= \sum_k (\delta_{pk} f'(s_k) w_{jk})\, f'(s_j) a_i \tag{9}$$

Defining

$$\delta_{pj} = \sum_k \delta_{pk} f'(s_k) w_{jk}, \tag{10}$$

we arrive at a recursive formula for propagating error in output back through the network. Take

$$\Delta w_{ij} = \eta \delta_{pj} f'(s_j) a_i \tag{11}$$

Essentially, this method for updating weights performs a localized gradient descent in error cost over weight space.

5 Auxiliary Error to Constrain Hidden Unit Activity

In order to perform dimensionality reduction we will provide one subset of units for each scalar to be encoded by the network. The activity in each such subset of units must display scalarized behavior; such a subset of units is called a scalar-set. See figure 4. An m-dimensional surface embedded in an n-dimensional data space will have n scalar-sets in the input layer, m scalar-sets in the hidden layer, and n

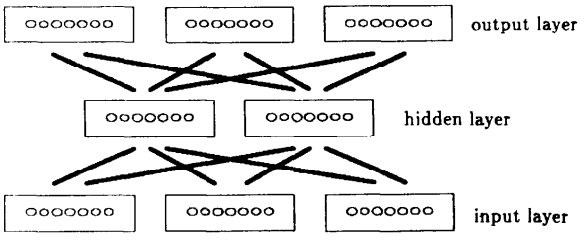


Figure 4. Network of scalar-sets for performing dimensionality-reduction. Heavy lines between scalar-sets indicate weight connection between every unit in each scalar-set.

scalar sets in the output layer. The training procedure will consist of repeatedly presenting data samples to the input layer, observing activity at the output layer, and updating weights in the network according to equations (6) and (11).

If all of the data presentations to scalar-sets at the input layer conform to scalarized representations for the scalar components of the data vector, then after a suitable training period the output will come to faithfully mimic the input. Unfortunately, there is no guarantee that the hidden units will adopt the scalarized representation in their transmission of activity from the input layer to the output layer. In general their coding behavior will be unpredictable, depending upon the initial randomized state of the network and the order of data sample presentation.

What is needed is a way to force the scalar-sets in the hidden layer into adopting the prescribed scalarized pattern of activity. For this purpose we introduce an auxiliary error term, γ_j, to be added to the error in activity at the hidden layer, δ_j, which was propagated back from error in activity at the output layer (10). The weights connecting the input layer and the middle layer are now updated according to

$$\Delta w_{ij} = (\delta_j + \gamma_j) f'(s_j) a_i \tag{12}$$

γ must be of a suitable nature to pressure the hidden units into becoming scalarized as the training proceeds. We compute a set of γ_j for the units of every hidden layer scalar-set independently, as follows:

We may view the activity over the set of units in a scalar-set as the transformation, by the smearing function, S, of some underlying "liklihood" distribution, $p(j)$, over values in the interval, $0 < j < 1$. The activity in a scalar-set is the convolution of the liklihood distribution with the smearing function, sampled at every unit. Scalarized activity occurs when the underlying distribution is the Dirac Delta function. The strategy we suggest for adding auxiliary pressure to the scalar-set activity is simply to encourage scalarized behavior: add some factor to sharpen the peaks and lower the valleys of the liklihood distribution, to make it more like the Delta function. A convenient way of doing this is to raise the underlying distribution to some positive power, and normalize so that the total area is unity. In the general case, if this procedure were repeatedly applied to some distribution, one peak would eventually win out over all others. The procedure is summarized by the following equation:

$$\gamma(j) = \left(S * N\left\{\left[S^{-1} * a(j)\right]^q\right\}\right) - a(j) \tag{13}$$

The activity in the scalar-set, $a(j)$ is deconvolved with the smoothing function, S, to reveal the underlying liklihood distribution. This is raised to the power, $q > 0$, and then normalized (by N). This new underlying liklihood is now convolved with the smoothing function, S, and γ is taken as the error between this result and the current activity in the scalar-set.

Now, on every training trial the weight updating term, δ_j, pressures hidden units to adopt activities that will reduce the error between input layer activity and output layer activity, while the auxiliary error term, γ_j, pressures hidden units to adopt scalarized activity. In reality, a tradeoff parameter is introduced in equation (12) to weight the relative pressures from δ and γ.

In an actual implementation, $a(j)$ is not a continuous function, but rather consists of the activity in the finite, usually small, number of units in the scalar-set. Therefore the bandwidth available for conveying the underlying liklihood, $p(j)$, is small; sharp peaks in $p(j)$ are not representable because high spatial frequency information cannot be carried in a. An alternative expression for γ has been found acceptable in practice:

$$\gamma_j = N\left[S * a_j^2\right] - a(j) \tag{14}$$

Square the activity in each unit, convolve this squared activity in the scalar-set with the smearing function, S, then normalize so that the total activity in the scalar-set sums to a constant. This procedure for generating γ_j approximates the effect of encouraging scalarized patterns of activity in the scalar-set.

6 Sculpting the Energy Landscape

As noted above, the network training procedure carries out gradient descent. Weights are updated at each training presentation so as to reduce the energy cost, E. This cost is high when activity in the output layer differs from activity in the input layer, and, due to the auxiliary error term, γ, the cost is high when activity in hidden layer scalar-sets does not conform to the characteristic scalarized representation for scalar numbers. If, as is usually the case, no prior knowledge of constraint operating upon the data source is available, the network is initialized with random values for all weights, and E will be large at the outset.

Simple gradient descent methods commonly suffer from the problem that there may exist local minima in the energy landscape that are far from the global minimum. Once a network falls into a local minimum there is no escape.

The local minimum phenomenon has been reported by Rumelhart, et al [1985], in normal binary-variable connectionist training, where the only pressure to adjust weights comes from error between output and input activity. It should perhaps not then be surprising to encounter local minima in the dimensionality reduction problem, where we impose an energy cost factor due to non-scalarlike behavior in hidden units, in addition to normal cost for output activity deviation from input. In effect, what we are doing is adding two energy landscapes to one another. The weight adjustment that reduces one energy cost component may raise the other. Figure 5 is a simple illustration of one way in which adding two energy landscapes can create local minima.

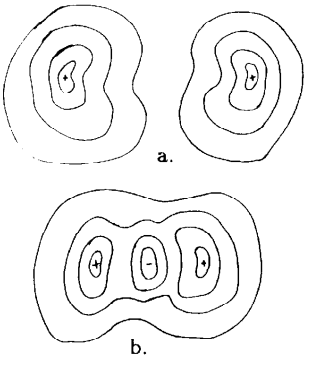


Figure 5. a. Neither of the energy potentials whose contours are shown has local minima by itself. b. But if they are moved near one another and added, local minina can be created in the resulting landscape.

Two strategies have been proposed for surmounting the local minimum problem. One is simulated annealing in a Boltzmann machine [Kirkpatrick, et al, 1983; Hinton, et al, 1984]. Briefly, simulated annealing allows the training process to probabilistically adjust weights so as to *increase* energy cost. This allows the procedure to jump out of local minima in energy. Boltzmann machine learning can be slow, and it requires certain conditions on the shape of the energy landscape in order to have a good chance of working. We have not investigated its applicability to the dimensionality-reduction problem.

Another strategy for skirting local minima involves changing the shape of the energy landscape itself as training proceeds. The idea is to introduce a parameter that makes the landscape very smooth at first, so that the network may easily converge to the local (and global) minimum. Then, gradually reduce this parameter to slowly change the landscape back into the "bumpy" energy potential whose minimum defines the network behavior actually desired. A variant on this technique has been used by Hopfield and Tank [1985] to train networks to find good (but not optimal) solutions to the traveling salesman problem (see also [Koch, et al, 1985]).

For the dimensionality reduction problem we take as the energy landscape smoothing parameter the parameter, ω, of the smearing function, S_ω. At the beginning of a training session, the activity in all scalar-sets describing scalar-valued numbers is smeared across virtually all of the units within each scalar-set. Figure 3c illustrates the activity across a scalar-set under a variety of smoothing parameters, ω.

This strategy creates two related effects. First, it reduces the precision to which the data values presented as input activity, and sought by the output error term, are resolved. Thus, local kinks and details of any constraint curve constraining the input data are blurred over more or less, depending upon ω. Second, under smearing with a large ω, auxiliary error on the hidden layers pressures each unit's activity to be not too different from its neighbor's activity. The activity in hidden unit layers is thereby encouraged to organize itself into adopting the scalarized representation.

Training begins with the smearing parameter, ω, set to a high value. The parameter is gradually reduced to its final, highest resolution smearing value according to a training schedule. Typically several thousand data-sampling/weight-updating trials are performed for each of five intermediate values for ω.

7 Performance

To date, this dimensionality-reduction method has been tested for cases where the input data is two dimensional, but constrained to lie on a one-dimensional curve ($n = 2, m = 1$), and where the input data is three-dimensional, but constrained to lie on a two-dimensional surface ($n = 3, m = 2$).

Figure 6 illustrates the underlying constraint curve for an $n = 2, m = 1$, test case. The X's represent locations indicated by output activity computed by the network when the input is drawn from points on the constraint curve. The extent to which X's lie on the curve simply demonstrate that network output conforms to input. Circles represent network output when scalar values are injected into the hidden layer. (This is done only to evaluate network behavior and is not part of the training procedure). The number next to each circle indicates the scalar value presented to the hidden layer.

Figure 7 illustrates the $n = 3, m = 2$ case in a similar way. Figure 7a is the true underlying constraint surface. Figure 7b represents network output for input data drawn from the constraint surface. Figure 7c illustrates network output when successive (u, v) pairs, $0 < u < 1, 0 < v < 1$, are injected directly into the hidden layer.

Note in figure 6 that the constraint surface is found successfully even though it doubles back on itself in both the x and y dimensions. In general, the more units presented

to a scalar-set, the better resolution available for capturing a constraint curve. Analysis of the behavior of individual units in a hidden layer scalar-set indicates that each unit in the hidden layer typically encodes a locally linear approximation to one portion of the constraint curve.

Some types of constraint curve cannot be discovered by this procedure. These are curves that double back on themselves radically. Figure 8 illustrates. The reason for this failure is that points such as a. and b. in figure 8 appear indistinguishable to the network early in the training procedure when S_ω causes very heavy smearing of their coordinate representations. They are therefore assigned similar encodings in the hidden unit layer. As ω is decreased, later in the training procedure, the network remains stuck in a local minimum of trying to encode both a. and b. using nearby hidden scalar values, when in fact it turns out that they are on opposite ends of the constraint curve and so should be assigned very different encodings in the hidden scalar-set. The energy landscape sculpting strategy does not work when, as the landscape smoothing parameter is decreased, the global minimum in energy potential suddenly appears in a very different location in weight space from where the previous local minimum had been.

A training protocol of 2000 trials for each of five values of the smearing parameter, ω, takes approximately 30 minutes for the $n = 2, m = 1$, case, with resolution of eight units per scalar-set, on a Symbolics 3600 lacking floating point hardware.

We anticipate that the extension of this technique to larger dimensionalities of input data, n, should be straightforward. The extension to greater dimensionality of the underlying constraint surface, m, remains somewhat uncertain and must be the subject of future research. It would also be important to develop some way to decide automatically what the appropriate constraint dimensionality, m, is for a given set of input data.

8 Conclusion

We have presented a mechanism for performing a dimensionality reduction type of abstraction over multidimensional data constrained to lie on a lower-dimensional surface embedded in the data feature space. A technique is given for representing in connectionist networks the scalar components of continuous vector-valued data. An auxiliary error pressure is introduced in order to pressure hidden units in the network into adopting this representation for scalar values.

This method has been shown capable of capturing a wide variety of underlying constraints implicit in data samples, despite the lack of any concise mathematical description of the constraint itself. Note that nowhere is the constraint curve described explicitly; its shape remains implicit in the weight connections in the network.

The network constructed by this method is able to use knowledge of constraint in order to encode information in a more concise representation than its original description as a data vector. We conjecture that such an abstraction mechanism may prove a useful building block for intelligent information processing systems.

Acknolwedgements

This work was carried out under the supervision of Professors Eric Grimson and Whitman Richards. I thank Dr. Jay McClelland and especially Aaron Bobick for valuable technical discussion. The presentation was much improved by comments from Jim Mahoney, Chris Atkeson, and Anselm Spoerri.

References

Ackley, D., Hinton, G., and Sejnowski, T., [1985], "A Learning Algorithm for Boltzmann Machines", *Cognitive Science,* 9, 147-169.

Hinton, G., Sejnowski, T., and Ackley, D., [1984], "Boltzmann Machines: Constraint Satisfaction Networks that Learn", Technical Report CMU-CS-84-119, Carnegie Mellon University.

Hopfield, J., and Tank, D., [1985], "Neural Computation in Optimization Problems", *Biological Cybernetics,* 1985.

Kirkpatrick, S., Gelatt, Sl, and Vecchi, M., [1983], "Optimization by Simulated Annealing", *Science,* 220, 671-680.

Koch, C., Marroquin, J., and Yuille, A., [1985], "Analog 'Neural' Networks in Early Vision", MIT AI Memo 751, MIT.

Kohonen, T., [1984], *Self-Orgainization and Associative Memory,* Springer-Verlag, Berlin.

Rumelhart, D., Hinton, G., and Williams, R., [1985], "Learning Internal Representations by Error Propogation", ICS Report 8506, Institute for Cognitive Science, UCSD.

Rumelhart, D., McClelland, J, PDP Research Group, [1986], *Parallel Distributed Processing: Explorations in the Structure of Cognition,* Bradford Books, Cambridge, MA.

Ullman, S., [1979], *The Interpretation of Visual Motion,* MIT Press, Cambridge, MA.

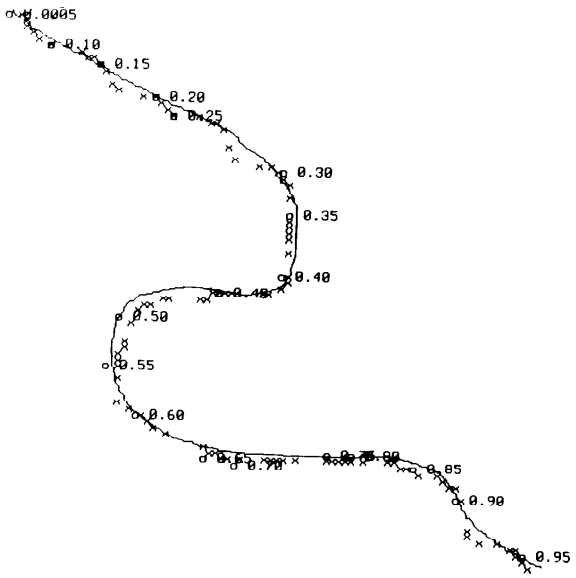


Figure 6. a. One-dimensional constraint in two-dimensional data. X's represent network output when input is taken from the constraint curve. Each circle represents network output when the hidden layer is injected with the scalarized representation of the number next to the circle. In this example scalar-sets were of size, 14 units.

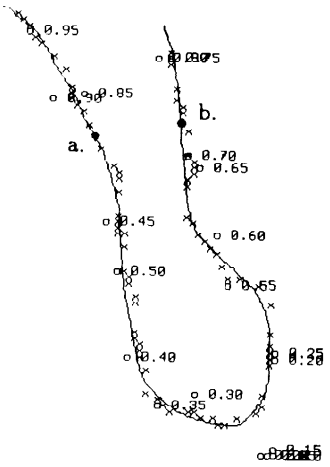


Figure 8. Failure of network training occurs when the constraint surface doubles back on itself sharply. Note how the number represented in hidden unit activity jumps around with respect to distance on the constraint curve.

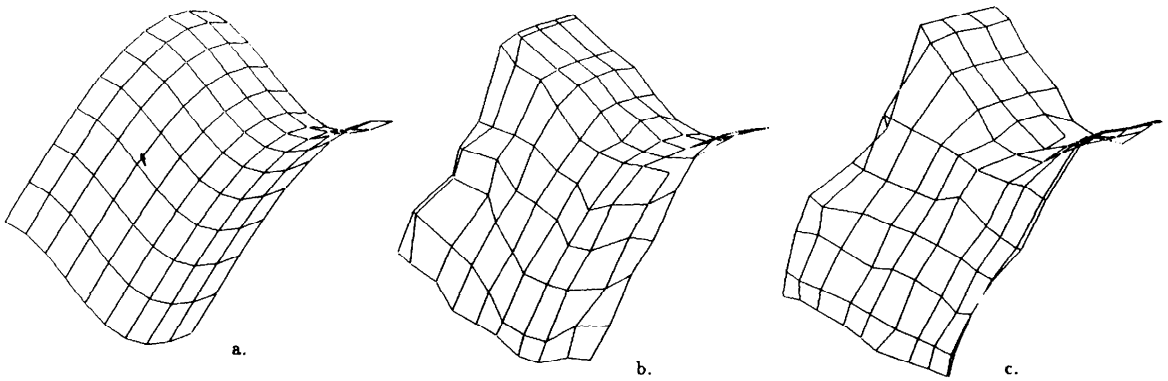


Figure 7. a. Two-dimensional constraint surface in three dimensions. b. Activity at output layer when input layer is fed data on the constraint surface. c. Activity at output layer when hidden layers are injected with scalar values between 0 and 1. In this example scalar-sets were of size, 14 units.