Complex Systems **3** (1989) 185–207

The Choice Problem: Neural Network Learning, Generalization, and Geometry

Alan J. Katz
Dean R. Collins
Marek Lugowski
Texas Instruments Incorporated,
Central Research Laboratories, Dallas, Texas 75265, USA

Abstract. We study the learning and generalization capacity of layered feed-forward neural networks in the context of mappings of arbitrarily long bit strings into one of three ordered outputs. Many signal-processing applications reduce to this problem. We use the back-propagation learning algorithm to train the network. We describe these mappings in terms of collections of linear partitions of the input space and the state spaces of the hidden layers of neurons. This description accounts for the properties of the mappings and suggests that learning and generalization are enhanced by training with boundary points of the input space. Several examples are included. We close with implications for layered feed-forward networks in general.

1. Introduction

Learning and generalization are key properties of neural networks and underlie much of the interest in neural network computation [1]. In particular, supervised learning in the neural network domain involves associating a set of input vectors $\{A\}$ with some output set of vectors $\{B\}$, where $\{A\}$ and $\{B\}$ are not, in general, equivalent sets. Multilayer neural networks learn, in principle, any arbitrary mapping [2]. For discrete, binary input spaces, a single layer of hidden neurons proves sufficient for learning an arbitrary mapping. Continuous input spaces usually require two layers of hidden neurons to guarantee neural network learning unless the neurons generate nonlinear surfaces; in that case, a single layer of hidden neurons is enough. In many applications, only part of the mapping from the input set $\{A\}$ to the output set $\{B\}$ is known. In such cases, the neural network trains on the incomplete mapping. A neural network generalizes when a previously unseen input stimulates the desired output. Because a large number of mappings are consistent with a given set of partial data, generalization occurs in limited and very special cases. Generalization is similar to curve-fitting, where problems are often vastly underconstrained [2].

Feed-forward neural networks with linear-threshold neurons learn by linearly partitioning the input space into regions: each neuron defines a hyperplane that bounds and isolates some region in either the input space or the state space of other neurons in the network [3,4]. The input to a linear-threshold neuron is a linear function of the outputs of the other neurons; the output from the neuron is a bounded, nonlinear function of the input. Neural networks with nonlinear-threshold neurons divide the input space into regions with arbitrarily shaped boundaries; the input to these neurons is a nonlinear function of the outputs of the other neurons. Understanding mappings in feed-forward neural networks requires knowing the number, the topology, and the geometry of the regions defined by the inverse mapping of the output points. This geometrical approach to neural networks derives from the field of threshold logic devices [5] and earlier studies of the perceptron model [6,7]. The perceptron of Rosenblatt [6] and Minsky and Papert [7] includes no hidden layers of neurons and, therefore, only carries out simple, linearly separable problems.

In this paper, we apply the geometrical point of view of [3,4,7] to mapping arbitrarily long bit strings into one of three outputs. The problem involves concatenating short bitstrings, which could represent sensor outputs, and mapping these strings into ordered sets of one of three outcomes. We name the mapping the Choice Problem to emphasize connections to areas of application. Our aim is to provide a complete geometric description of a neural network problem that is simple and yet rich enough to illustrate the principles of layered feed-forward network computation. Such understanding establishes a framework for better algorithms and better network design.

We train the network with the back-propagation learning algorithm [8]. We demonstrate how to evolve from an "easy" to a "hard" mapping in the sense of Minsky and Papert [7]. The issue of generalization is related to determining an appropriate partition of the input space. Knowing such a partition then leads to an estimate of the minimum number of training vectors required to encode the mapping. We discuss the interplay among the number of hidden neurons, the number of partitions, and the representation of the input vectors. We study how learning and generalization degrade for noisy and incomplete data from the point of view of the geometry of the input space.

The organization of the paper is as follows. In section 2, we develop the geometrical point of view of neural network mappings. Section 3 contains a discussion of the Choice Problem and presents the relevant data. Conclusions follow in Section 4.

2. The geometry of mappings in feed-forward neural networks

In this section, we interpret general binary mappings in layered feed- forward networks in terms of linear partitions of the input space. This interpretation

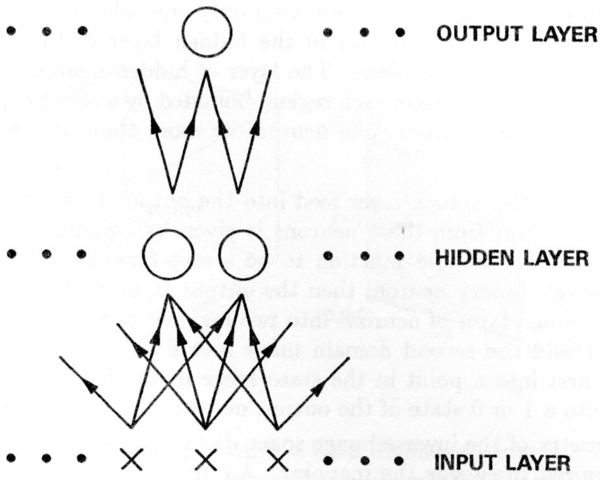

Figure 1: Architecture for a layered feed-forward neural network. Only points in neighboring layers interact.

provides insights into the notions of network learning and generalization.

Figure 1 shows a typical architecture for a layered feed-forward network with a single hidden layer. Each neuron is connected to every neuron in the previous layer, but connections between two neurons in the same layer and backward connections are not allowed. The ith neuron of the network performs the following function:

$$Y_i = f(\sum_j T_{ij} X_j - \Theta_i), \tag{2.1}$$

where T_{ij} is the connection strength between the ith and jth neurons, X_j is the input from the jth neuron, Θ_i is a threshold and is fixed for each site i, and f is some nonlinear function that commonly is sigmoidal in shape. A typical function f is $f(x) = 1/2(1 + \tanh x)$.

To simplify the analysis, we replace the threshold function f with the step-function or Heaviside function; the output Y_i from the ith neuron is then:

$$\begin{aligned} Y_i &= 1 \quad \text{if} \quad \sum_j T_{ij} X_j - \Theta_i > 0 \\ Y_i &= 0 \quad \text{if} \quad \sum_j T_{ij} X_j - \Theta_i \le 0. \end{aligned} \tag{2.2}$$

The equation $\sum_j T_{ij}X_j - \Theta_i = 0$ defines a hyperplane, which divides the input space into two regions; points on only one side of the hyperplane activate the neuron. Each neuron in the hidden layer of the network of figure 1 defines such a hyperplane. The layer of hidden neurons provides a partitioning of the input space; each region, bounded by a set of hyperplanes, activates none of the neurons, one neuron, or more than one neuron (see figure 2).

Outputs from the hidden layer feed into the output layer of neurons in figure 1. The output from these neurons is given by equation (2.1), where we again take the threshold function to be a step-function. If the output layer has a single binary neuron, then the output layer partitions the state space of the hidden layer of neurons into two regions: one domain maps into the output 1 and the second domain maps into 0. A region in the input space maps first into a point in the state space of the hidden neurons and ultimately into a 1 or 0 state of the output neuron.

The geometry of the inverse image space dictates how many hidden neurons are required to encode the mapping. An upper bound on the required number of hidden neurons is $n-1$ if the binary input space decomposes into n disjoint regions and any region is linearly separable from its complement.

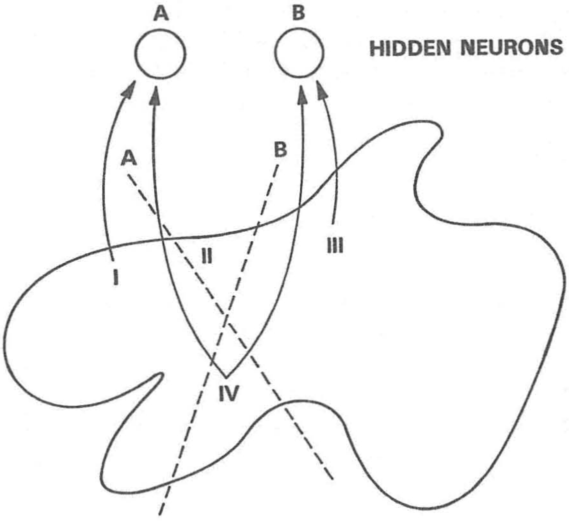

Figure 2: Partitions A and B are defined by neurons A and B respectively. Region I activates neuron A; Region III activates neuron B; Region IV activates both A and B, whereas Region II activates neither neuron.

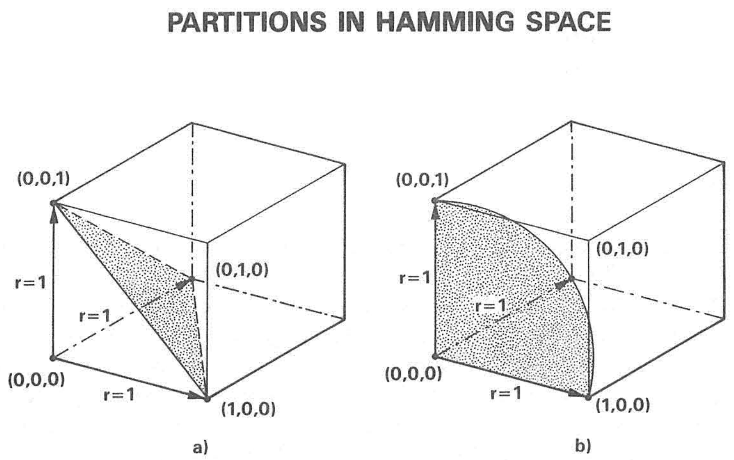

Figure 3: Points that are within a fixed Hamming Distance from the origin $(0, 0, 0)$ lie on (a) a hyperplane, or, equivalently, (b) within a hypersphere, as defined in the text.

An example of a linearly separable region in a binary input space is a hypersphere, which is the set of all points within a fixed Hamming radius of a vertex of the hypercube. To see that hyperspheres are linearly separable, we can consider hyperspheres about the origin $(0, 0, 0, \ldots, 0)$ without any loss of generality. For a d-dimensional input space (bit strings of length d), the hyperplane defined by the equation

$$\sum_{i=1}^{d} X_i = \Theta \tag{2.3}$$

where X_i denotes the ith bit and the threshold $\Theta = r$, clearly separates points within a Hamming distance r from the remainder of the space. We use the properties of hyperspheres to analyze the Choice Problem (see figure 3).

The output layer of neurons further divides the state space of the hidden layer into regions. If the output layer contains a single neuron, which specifies one partition, then the state space of the hidden layer must be linearly separable; otherwise, the network fails to encode the mapping. A linearly separable representation of the state space for the hidden layer always exists if we provide enough neurons [2]. This claim is clearly true when $n - 1$ hyperplanes suffice to partition an input space with n disjoint regions. In that case, we arrange for each of $n - 1$ regions to activate a unique neuron and the last region to activate none of the hidden neurons. The state space of the hidden layer then includes the following n vectors, which are $n - 1$ bits long:

$$(0, 0, 0, \ldots, 0)$$

$$\begin{array}{c}(1,0,0,\ldots,0)\\(0,1,0,\ldots,0)\\\cdot\\\cdot\\(0,0,0,\ldots,1).\end{array}$$

This representation is linearly separable, no matter how the vectors map into the output. The nonzero vectors in the above list define a hyperplane in an $n-1$ dimensional space, since they form a hypersphere about the origin. By letting the hyperplane intersect the line segments between the nonzero vectors and the origin either at a distance $1-\varepsilon$ or at a distance $1+\varepsilon$ from the origin ($\varepsilon \ll 1$), we achieve any desired partition of the state space.

A similar argument applies when more than $n-1$ hidden neurons are needed. We first subdivide the input space into hyperspheres; this can always be done, though such a procedure does not necessarily lead to the most efficient representation of the mapping. Since one hyperplane isolates a hypersphere, a hypersphere is linearly separable from the remainder of the space. We then assign one hyperplane to one hidden neuron. From this point on, the argument is the same as the earlier one.

These arguments generalize when the output layer contains several neurons. The state space no longer needs to be linearly separable, since other partitions of the state space come into play. But for n output states we must be able to divide the state space into n regions, each linearly separable.

We need to relax our assumption that the threshold function f is a step-function. In back-propagation, f is sigmoidal and varies smoothly between 0 and 1 [8]. For continuous threshold functions the partitions are rounded and the boundaries between regions are no longer sharp [2]; nevertheless, the above conclusions hold as long as well-defined regions remain. The mapping of points near a boundary, however, can be ambiguous.

The geometry of the inverse mapping depends on the representation of the input data. Changing representation changes the encoding of the mapping. An increase in the dimensionality of the input space often simplifies the mapping. The *XOR* problem [8] in two dimensions, for example, is linearly separable in three dimensions.

The above picture of feed-forward neural networks offers insight into generalization. To generalize, the network first learns an appropriate partitioning of the input space. If the mapping requires m partitions, then a network with at least m hidden neurons suffices. We must train the network with enough points to ensure the m partitions separate the appropriate regions. In a d-dimensional input space, $d \times m$ independent training vectors uniquely define m hyperplane partitions. In a discrete input space, the hyperplanes lie between points of the space, and we cannot rigidly constrain the positions of the hyperplanes. We then train the network, where appropriate, with $2 \times d \times m$ points, which define hyperplanes on both sides of every boundary and constrain the partitions to lie between the boundary points. These observations are useful when the mapping is known, and we wish to expedite learning (see figure 4).

Knowledge of mappings is often incomplete. When this happens, networks necessarily fail to generalize. If the geometry of the input space is unknown, then the number of hidden neurons is unconstrained. The smaller the number of hidden neurons, the more the network tends to cluster points and map them into the same output. In many applications, clustering is preferable to splitting points [9]. But limiting the number of hidden neurons affects the learning rate [10], so there is a trade-off between generalization and learning efficiency.

The role of noise and incomplete data in neural network performance is clear in the context of the geometry of the input space. We assume the neural network trains on error free data. The effect of corruption of the input data on neural network generalization and learning grows as the number of independent regions in the input space increases. The ratio of the number of boundary points to the total number of points in the input space is a rough measure of the susceptibility of the mapping to error at small error content. We say more about the effect of errors on mappings in the next section.

BOUNDARY POINT CONSTRAINTS

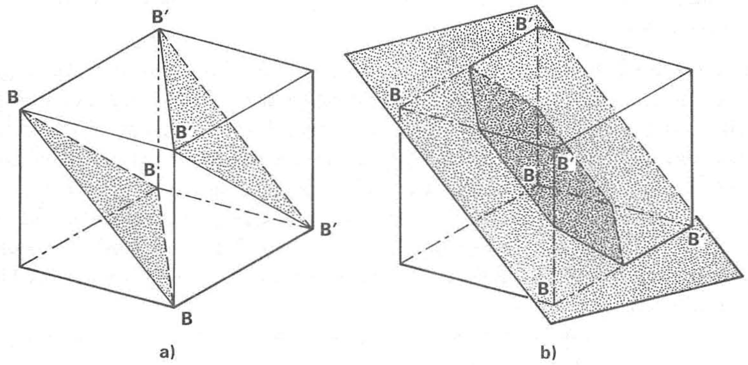

Figure 4: Boundary points constrain position of hyperplane. Points marked with B and B' indicate boundary points for two clusters, which are hyperspheres of radii 1 about the unmarked vertices. Two planes indicated in (a) constrain the position of partition shown in (b).

3. The Choice Problem

We apply the results of the last section to an example problem. We wish to train a neural network to monitor a system or an operation. The network receives input from N sensors, which track various aspects of the system; the number N is arbitrary. A sensor outputs one of three states: (+), (0), or (−), where (+) denotes an acceptable rating, (−) signifies an unacceptable rating, and (0) indicates an uncertain rating and warns of potential failures of the system. We combine the outputs from the N sensors and feed them to a neural network, which maps the string of sensor outputs to a (+), (0), or (−) state. The network is trained to choose a final state based on the sensor data.

To illustrate this Choice Problem, we assume the sensors measure the status or the performance of a car: the sensors may track oil pressure, engine temperature, fuel level, tire pressure, battery charge, mileage, tread wear, or any of a number of possible variables. At any point in time, each sensor registers a (+), (0), or (−) state. If all sensors output a (+), then the car performs satisfactorily, and the neural network should map this state into (+). On the other hand, if any of the functions is unacceptable, then the neural network should choose a (−).

We then define the initial Choice Problem, which we designate case A, as follows:

1. If any sensor output is in a (−) state, then the final output state is also (−).
2. If no sensor yields a (−) state, then the final state is (+) [(0)] when a majority of the sensors output $(+)'s$ $[(0)'s]$. We use odd numbers of sensors to avoid ambiguity.

The nature of the mapping can be more subtle. For example, we might include a sensor to measure the road condition (is it wet, icy, rough, or bumpy ?). A (0) status for tread wear combined with a (+) rating for all the other variables may rate no worse than a (0) in the final output. But if the state of the road condition is also (0), then the combination of a (0) rating for both tread wear and road condition may lead to a (−) output state. As the nature of the mapping becomes higher-order, learning times for the neural network generally are longer [11]. We consider an example (case B) of this more complex mapping below.

We use a bit-string representation for the input states and the output states, in analogy to a gauge on a car:

$$\begin{array}{ccc} (-) & \longrightarrow & (0\ \ 0\ \ 1) \\ (0) & \longrightarrow & (0\ \ 1\ \ 0) \\ (+) & \longrightarrow & (1\ \ 0\ \ 0). \end{array}$$

This representation has several advantages. For the N-sensor problem, the inverse mapping of the output state (−) occupies a hypersphere that has

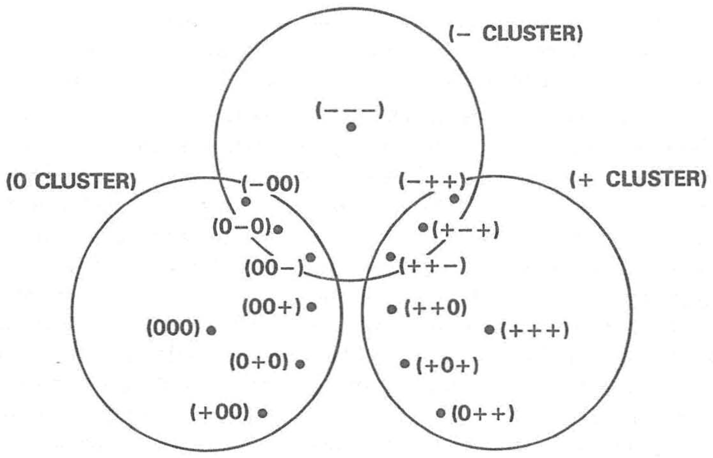

Figure 5: Representation of clusters for three-sensor Choice Problem.

radius $2(N-1)$ in the Hamming metric (which derives from comparing the states of individual bits) and is centered about the point $((-)(-)\ldots(-))$ where each sensor assumes a $(-)$ state. Many points within this hypersphere never appear in the mapping, and they are neglected for now (a total of $3N$ points are relevant to the mapping). A single hyperplane bounds the surface of the hypersphere and separates this set from the remainder of the space. We also must isolate points that map into $(+)$ from those that map into (0). A second hyperplane separates these two sets since the inverse mappings of the two output states, (0) and $(+)$, cluster into disjoint hyperspheres (if we ignore $(-)$ states) about the points $((0)(0)\ldots(0))$ and $((+)(+)\ldots(+))$, respectively (see figure 5).

We need two hyperplanes and, consequently, two hidden neurons to realize the mapping. In case A we can, however, dispense with the hidden layer, since the output layer alone, which contains three neurons, can suitably encode the mapping. Since we will consider more complicated mappings, which do require hidden neurons, we include a hidden layer to provide a basis for comparison. Moreover, back-propagation, which we use for our computer simulations, finds solutions by gradient-descent and gets stuck in local minima [8]. Adding hidden neurons eliminates local minima and increases the number of ways to realize the mapping.

Figure 6 shows the learning efficiency for three sensors (9-bit sensor data) as a function of the number of neurons in the hidden layer and the average number of presentations of vectors in the training set. We use a three-layer feed-forward neural network, and we train it by back-propagation. Each data point represents the average of five runs. The simulations are performed on the Neural Network Workstation of Texas Instruments [12]. The algorithm returns output in analog form: each of the three output nodes or neurons assumes a value between 0 and 1. We take an output neuron to be in a 1 state if its analog value is > 0.5 and in a 0 state if its analog value is < 0.5.

There are 27 vectors to learn. The learning parameters are the same for all simulations in this study. The smoothing parameter, which corresponds to α in the learning equations of Chapter 8 of [8] and controls the degree of over-relaxation, is set to 0.9. The rate of learning (η in [8]) is 2.0 in these simulations. We also include a speed of correction parameter, which we set to 1; the speed of correction indicates how many errors are found before the changes are applied to the weight matrix. The order of presentation of the training vectors is random. As figure 6 shows, network learning is complete with four hidden neurons and at least 300 trials/vector. A hidden layer with a single neuron defines only one hyperplane and cannot divide the input space into more than two regions. The network also learns this simple mapping with no hidden neurons, but an average of 400 presentations/vector is required to achieve 100% accuracy.

Adding hidden neurons increases the number of ways to realize the map-

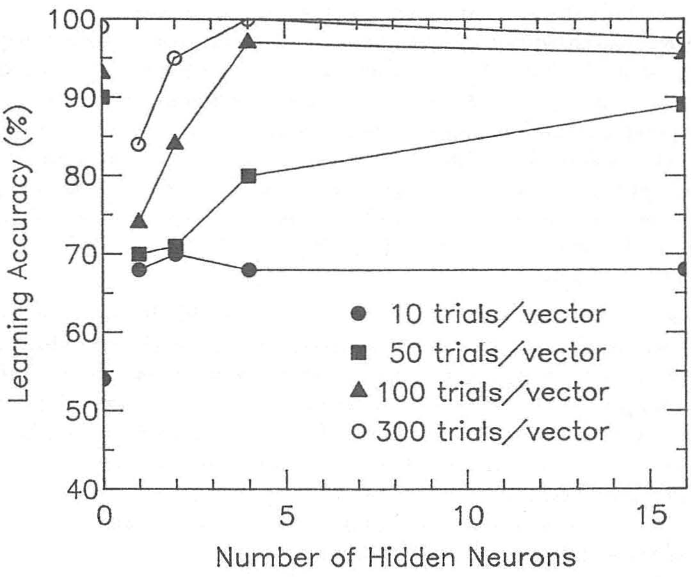

Figure 6: Learning accuracy as a function of number of hidden neurons and the average number of trials or presentations per training vector. There are 27 vectors in the training set; the order of presentation is random.

ping. As the number of hidden neurons grows in figure 6, the learning efficiency is enhanced. When the average number of presentations per vector falls below 300, however, increasing the number of hidden neurons does not fully compensate for the insufficient number of training trials. At 10 trials/vector the addition of hidden neurons has no effect. As the number of hidden neurons increases from four to sixteen, the learning accuracy systematically appears to decrease for 100 and more trials/vector. This trend is within the statistical variance of the simulation, however.

We next increase the complexity of the problem by altering the definition of the Choice Problem: in case B the point $((-)(-)(-))$ now maps into $(+)$ instead of $(-)$, but the rest of the mapping remains the same. The learning parameters take the same values they did in case A. The mapping has the character of the *XOR* problem, since two disjoint sets of points produce the same output point. This mapping is not possible without the use of hidden neurons. Figure 7 shows the training efficiency for all points in the mapping, except for $((-)(-)(-))$, as a function of the number of copies and the total number of presentations of $((-)(-)(-))$; for a particular abscissa value in figure 7, the average number of presentations of the other 26 vectors is simply the abscissa divided by the number of copies of $((-)(-)(-))$ in the training set. Each data point represents an average of seven simulation runs; four hidden neurons are used. The recognition accuracy for the set of 26 vectors decreases with increasing number of copies of $((-)(-)(-))$ in the training set. If only one copy of $((-)(-)(-))$ appears, the network readily learns to map the 26 vectors in the training set. The corresponding state of the network (that is, the matrix of weights) is at a relatively deep local minimum of the energy function (here measured in terms of distances between desired and actual outputs). The probability of finding the global minimum (all 27 states learned) is small, however (see figure 8).

The situation changes if we weight the training procedure in favor of the point $((-)(-)(-))$. Both the asymptotic limit of the learning accuracy and the rate of learning decrease fairly steeply with increasing number of copies of $((-)(-)(-))$. Increasing the number of copies can enhance the learning of the complete mapping (all 27 vectors), but adding too many copies may produce new minima in the energy function and impede the convergence of the network to the desired state. In figure 8, we show the learning accuracy of the mapping of the point $((-)(-)(-))$ alone as a function of the total number of presentations of $((-)(-)(-)) \rightarrow (+)$ during learning. The family of curves represents, as in figure 7, results for training sets with varying numbers of copies of the exceptional case. The recognition accuracy of the exceptional case increases with number of training trials and number of copies. For a fixed number of training trials the learning accuracy increases rapidly with the number of copies of the exceptional case in the training set.

Figure 9 illustrates what happens to learning when the network first trains on the 27-vector training set and then trains on the exceptional case alone. The network forgets the original mapping exponentially fast and maps all vectors into $(+)$, the output state of $((-)(-)(-))$. Figure 10, on the other

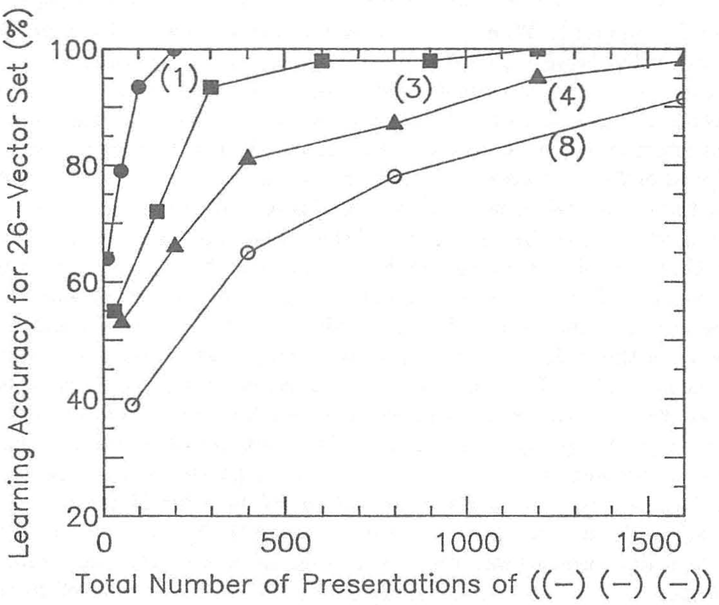

Figure 7: Learning accuracy for 26 vectors (the point ((−)(−)(−)) is excluded) in the three-sensor Choice Problem as a function of the total number of presentations and of the number of copies of the vector ((−)(−)(−)) in the training set. The vectors are presented in random order; there are four hidden neurons. The numbers in parentheses denote the number of copies of ((−)(−)(−)) in the training set.

hand, shows how rapidly the network learns the exceptional case when it trains on that vector alone.

To test how well the network generalizes case A, we train the network on subsets of the input list and test on vectors not previously seen. Figure 11 plots the recognition accuracy for datapoints not previously seen versus the number of vectors in the training set for the three-sensor problem. Vectors in the training set are chosen randomly; the hidden layer contains four neurons. Each point in figure 11 represents an average over different training sets and over initial starting conditions for a particular training set.

The results reflect peculiarities in both the Choice Problem and the back-propagation learning algorithm. When the network trains on a single vector, the recognition accuracy is around 0.50. This high recognition accuracy reflects two facts: (1) the majority of vectors (19 out of 27 in the three-sensor problem) map into (−) and (2) the back-propagation learning algorithm trains the network to map everything into the output state given by the training vector. We then calculate a recognition accuracy of 0.49 from the probability of choosing to train with a vector that maps into (−), which is 0.7, times the recognition accuracy for that case, which is again 0.7. When the network trains with no vectors, the recognition accuracy drops to zero because the output from the network is always ambiguous.

We improve the ability of the network to generalize by training only with vectors that fall along the boundaries between regions of the input space. These points are the most difficult ones for the network to learn. Once it learns these points, the positions of the hyperplanes are constrained to lie along the boundaries between regions, and the network generalizes. In section 2, we estimated the minimum size of the training set: $M_{est.} = 2 \times d \times m$, where d is the dimensionality of the space and m is the number of partitions (hyperplanes). For the N-sensor problem $d = 3 \times N$ and $m = 2$. In practice, our approximation overestimates the minimum size of the training set: some of the boundary points constrain more than one hyperplane. Figure 12 shows the number of vectors we used to learn the mapping for cases $N = 3$, 5, and 7 and the theoretical curve: $M_{est.} = 2 \times d \times m$. The total number of

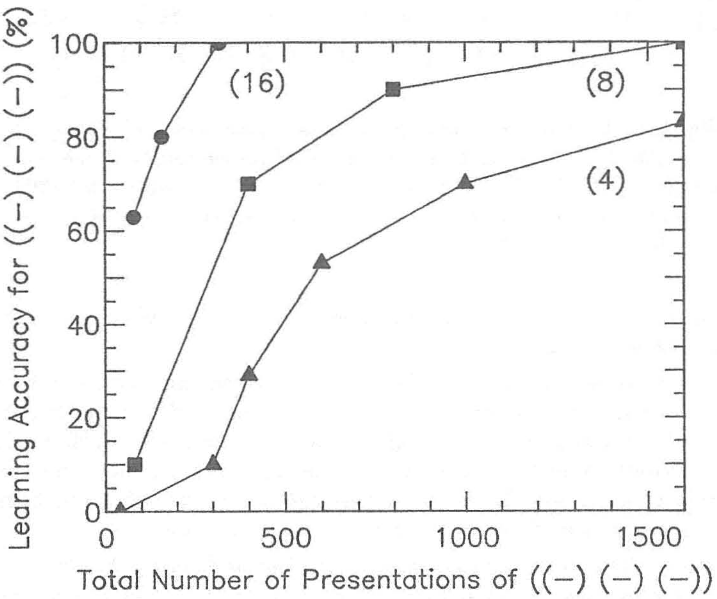

Figure 8: Learning accuracy of the mapping $((-)(-)(-)) \rightarrow (+)$ as a function of the total number of presentations and of the number of copies in the training set of the point $((-)(-)(-))$. The vectors are presented in random order; there are four hidden neurons. The numbers in parentheses denote the number of copies of $((-)(-)(-))$ in the training set.

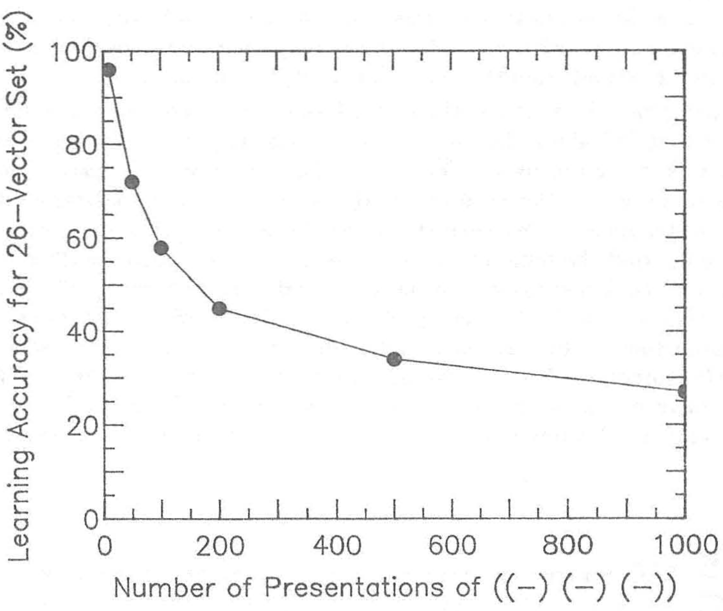

Figure 9: Learning accuracy for 26 vectors (the vector $((-)(-)(-))$ is excluded) as a function of the number of presentations of the vector $((-)(-)(-))$. The network first trains on the 27-vector set (300 presentations/vector); then it trains on the mapping $((-)(-)(-)) \rightarrow (+)$ alone.

points in the input space scales exponentially with N, whereas the size of the training set scales linearly.

Figure 13 shows how rapidly the network performance for the five-sensor problem degrades as we randomly remove vectors from the hand-chosen training set. The initial training set is the same as the one used in figure 12 for $N = 5$. Network performance does not degrade rapidly until the number of boundary points falls below 30. These results indicate that our estimate of the minimum size of the training set is not a strict one.

In many situations, the data are incomplete and noisy. Figure 14 summarizes how well the network generalizes for $N = 3$, 5, and 7 as a function of the noise content. To generate noisy data, we change each bit of an input data string to its complementary state with probability Φ. We train, however, on noise-free data. The recognition rate for noisy data increases with N for most values of the probability Φ. This trend is understandable. At smaller values of Φ, the ratio of points near the boundary to interior points controls the susceptibility of the mapping to errors: the smaller this ratio, which decreases with N, the greater the error tolerance should be. As Φ

approaches 1, on the other hand, the fraction of points that map into the correct state approaches 1 asymptotically with N.

To understand the results of figure 14 in more detail, we calculate the performance of the network for $\Phi = 1$: each bit changes its state with probability 1. Our calculation depends on how the input space divides into distinct regions. As an example, we take the three-sensor case and assume the network "learns" to isolate all points within a Hamming radius of five (for the N-sensor problem, the radius is $2N-1$) of the point $((-)(-)(-))$ from the other points in the space. A second hyperplane further separates points that map into $(+)$ from those that map into (0). For this realization of the mapping, only points that are a Hamming distance of four from $((-)(-)(-))$ map into the correct output. This implies a recognition rate of 0.44, which agrees with the experimental result. For $N = 5$ and $N = 7$, we calculate recognition rates of 0.67 and 0.77, respectively, which do not agree with the experimental results.

To explain this discrepancy, we illustrate with the five-sensor problem a second way to realize the Choice Problem. The dimensionality of the input

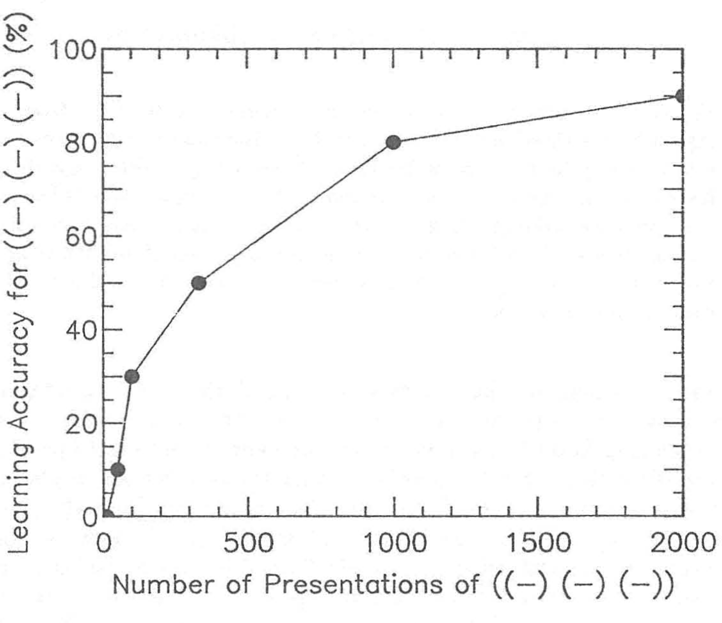

Figure 10: Learning accuracy of the case $((-)(-)(-)) \rightarrow (+)$ as a function of the number of presentations of the vector $((-)(-)(-))$. The network first trains on the 27-vector set (300 presentations/vector); then it trains on the mapping $((-)(-)(-)) \rightarrow (+)$ alone.

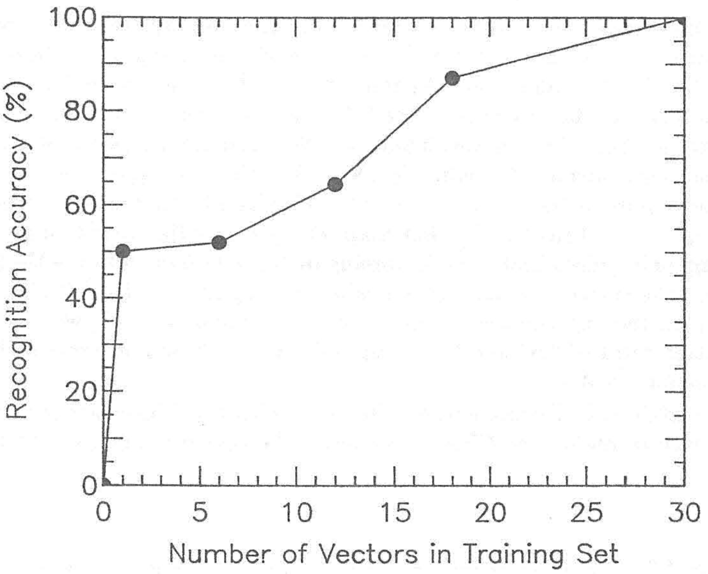

Figure 11: Recognition accuracy versus number of vectors in the training set for the three-sensor Choice Problem. The training sets are chosen randomly from the complete list of 27 vectors (no vector repeats, however). Each point in the figure represents an average over at least four different training sets and over five initial weight configurations. On average, each vector in the training set is presented around 1000 times. The order of presentation is always random. The hidden layer contains four neurons.

space for the five-sensor Choice Problem is 15 and, therefore, 15 independent points define a hyperplane in this space. Using this fact, we isolate the 16 points that map into $(+)$ with one hyperplane and isolate the 16 points that map into (0) with a second hyperplane. With these partitions in place, only points at a Hamming distance between four and eight from $((-)(-)(-))$ continue to map into the correct state at $\Phi = 1$. We now calculate a recognition rate of 0.82, which is more in line with the simulations. Similar arguments carry over to higher-dimensional problems. These calculations and simulations demonstrate how ambiguous network performance is unless we impose enough constraints to guarantee a unique realization of the mapping.

The results in figure 14 are dominated by the fact that as $N \rightarrow \infty$ the fraction of states that maps into either $(+)$ or (0) becomes negligibly small. In figure 15, we plot the recognition accuracy versus Φ for only those states that map into either $(+)$ or (0). The seven-sensor case is shown. The theoretical curve in the figure is given by the function $f(\Phi) = (1-\Phi)^7$, which is the probability to flip at least one of the seven bits that signal a $(-)$ state.

All mappings up to this point derive from the simple rules described earlier (case A mapping). We next study network learning for random mappings (case C). A random mapping assigns outputs randomly to each input vector. Such a mapping cannot be written in terms of a simple algorithm. More generally, a mapping is random if the shortest algorithm that generates the mapping exceeds some threshold length [13]. The set of random mappings is an important class of mappings and includes many pattern-recognition problems.

Random mappings are hard for feed-forward neural networks to learn because of the geometry of the inverse mapping of the output space. To

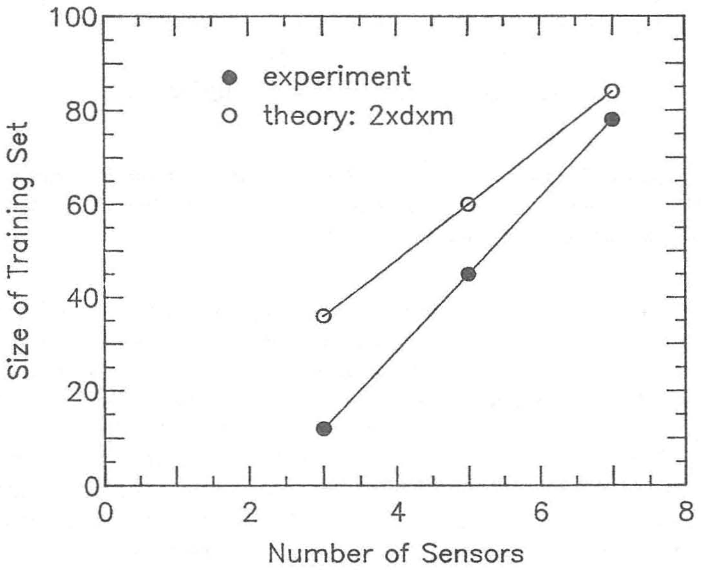

Figure 12: Size of training set as a function of the number of sensors to obtain essentially complete recognition. The points in the training sets are hand-chosen to lie at the boundaries between regions and to be noncolinear. The average number of presentations/vector in the training set is around 400, and the order or presentation is random; the hidden layer contains four neurons. For the three- and five-sensor cases, the recognition accuracy is 100%; it is greater than 99.9% for the seven-sensor case. The theoretical curve is a plot of the function $M_{est.} = 2 \times d \times m$.

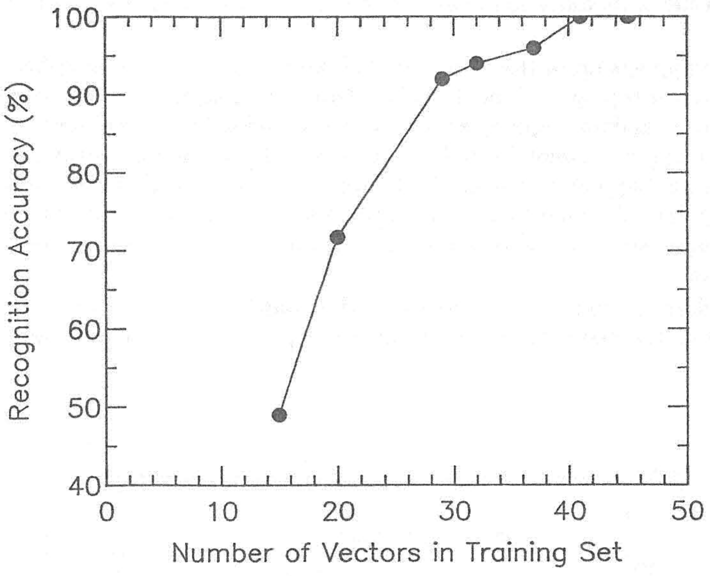

Figure 13: Degradation of network performance for the five-sensor choice problem. The simulation conditions and parameters are the same as the ones described in figure 12.

illustrate the problem, we examine the mapping from $R^2 \Rightarrow \{0,1\}$, where we represent R by a triangular lattice of points. We assume each point in the plane takes the value 1 or 0 with probability 0.5. This mapping defines a site percolation problem in two dimensions [14], which is well known to have a percolation threshold at 0.5. At the percolation threshold, the inverse mappings for both 1 and 0 are fractal geometries [15], which have structure at all length scales. To realize this mapping requires an infinite number of partitions as the lattice spacing or smallest length scale in the problem goes to zero or as we let the size of the input space go to infinity.

A key question then is how the average number of disjoint clusters of points and the geometry of these clusters scale with the dimensionality of the input space for the random mapping problem. For a random Boolean mapping of a discrete, high-dimensional input space, where input strings of 1's and 0's map with equal probability into 1 or 0, the average number of hyperplane partitions needed to realize the mapping with a feed-forward neural network grows with N, the dimensionality of the input space. Such a problem is hard in the sense defined by Minsky and Papert [7]. Learning times with the back-propagation learning algorithm for other such hard problems are known to scale exponentially with N [11].

In the Random Choice Problem, a fraction of the total points in the input space appears in the mapping, so the scaling problem is somewhat less severe. Within the subset of points of interest, no two points ever get closer than a Hamming distance of two. We therefore define two points to be contiguous in this discussion if their separation in the Hamming metric is two. We present an example of a random mapping for three sensors in table 1. How successfully a feed-forward neural network with four hidden neurons learns such a mapping relates both to the number of disjoint or noncontiguous sets formed by the inverse mappings of the output vectors and the geometry and topology of these sets.

The neural network fails to learn the example after an average of 1300 presentations per vector of the input set. The inverse mapping of (+) yields only one contiguous region, but decomposes into at least three linearly separable clusters. The inverse mappings of (0) and (−) yield three and four disjoint regions, respectively, and require at least that many hidden neurons. A network with four hidden neurons, therefore, cannot learn this mapping

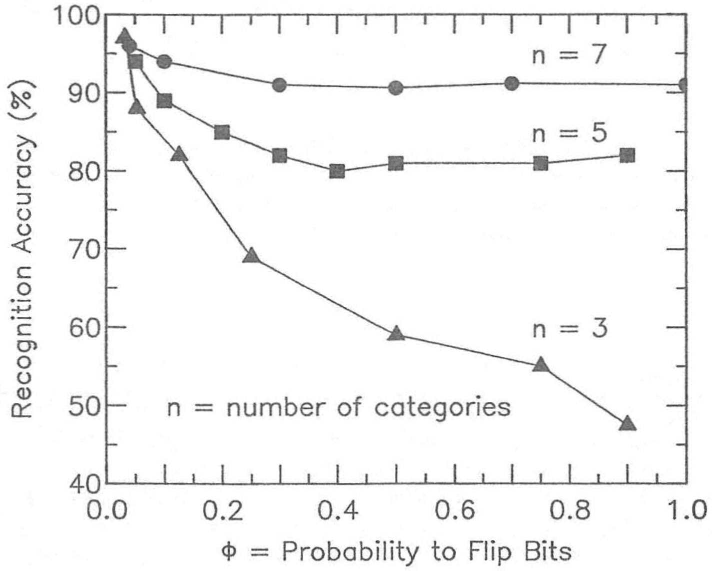

Figure 14: Generalization capacity with noisy data: results are shown for the $N = 3$, 5, and 7-sensor problems. Training is done with error-free data. The training sets are the same as the ones described in figure 12. Each vector in the training set is presented on average 400 times. The order of presentation is random. Testing is done on noisy data. Each point in the figure represents an average over ten test sets.

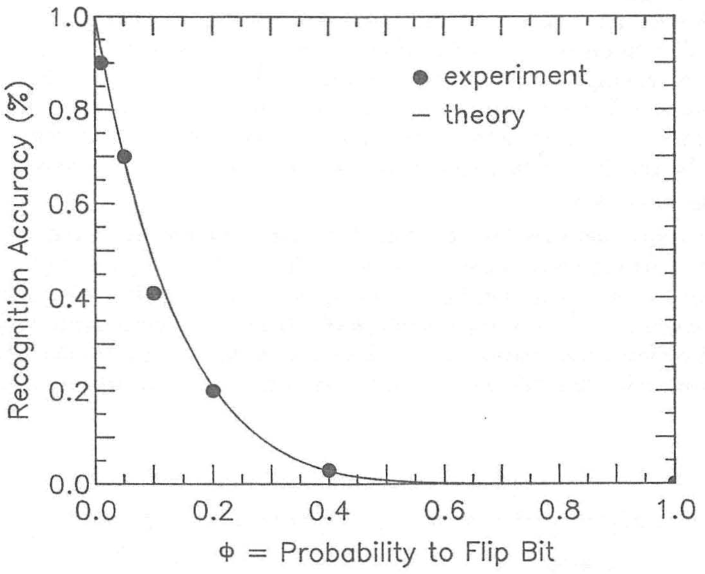

Figure 15: Generalization capacity with noisy data: only results for states that map either into (+) or (0) are plotted. Results are for seven-sensor problem. Conditions are the same as for figure 14. The theoretical curve is given by $f(\Phi) = (1 - \Phi)^7$.

Example		
$((+))(+)(+)) \Rightarrow (0)$	$((0)(+)(+)) \Rightarrow (+)$	$((-)(+)(+)) \Rightarrow (+)$
$((+))(0)(0)) \Rightarrow (+)$	$((0)(0)(0)) \Rightarrow (0)$	$((-)(0)(0)) \Rightarrow (0)$
$((+))(-)(-)) \Rightarrow (+)$	$((0)(-)(-)) \Rightarrow (-)$	$((-)(-)(-)) \Rightarrow (-)$
$((0))(0)(+)) \Rightarrow (+)$	$((-)(0)(+)) \Rightarrow (-)$	$((+)(0)(+)) \Rightarrow (0)$
$((0))(-)(0)) \Rightarrow (+)$	$((-)(-)(0)) \Rightarrow (+)$	$((+)(-)(0)) \Rightarrow (+)$
$((-))(+)(-)) \Rightarrow (0)$	$((+)(+)(-)) \Rightarrow (-)$	$((0)(+)(-)) \Rightarrow (0)$
$((-))(-)(+)) \Rightarrow (+)$	$((+)(-)(+)) \Rightarrow (+)$	$((0)(-)(+)) \Rightarrow (0)$
$((+))(+)(0)) \Rightarrow (+)$	$((0)(+)(0)) \Rightarrow (-)$	$((-)(+)(0)) \Rightarrow (+)$
$((+))(0)(-)) \Rightarrow (+)$	$((0)(0)(-)) \Rightarrow (+)$	$((-)(0)(-)) \Rightarrow (0)$

Table 1: An example of random choice problem for the three-sensor case.

without errors.

4. Discussion and conclusions

The Choice Problem demonstrates how the geometry of the inverse mapping relates to learning and generalization in neural networks. Feed-forward networks realize arbitrary mappings. But to obtain an arbitrary mapping, we must understand what partitions of the input space and the state spaces of the hidden layers give rise to the mapping. Once we do this, we gain insight into the degree of difficulty of the mapping, how to train the network in an optimal way, the sensitivity of the mapping to noise, the ability of the network to generalize the mapping, and the influence of representation on the mapping.

There are many ways to carry out a specific mapping. The number of ways depends on the architecture of the network. For example, the number of hidden neurons in the network controls the number of partitions and the size of the state space of the hidden layer. Characterizing a mapping in terms of a single realization of the mapping is meaningless. Instead, properties of the mapping should be expressed as averages over an ensemble of realizations of the mapping.

The nonrandom and random Choice Problems represent two extremes. The nonrandom problem is straightforward to realize and does not require additional neurons as we increase the number of input sensors. On the other hand, the random case requires more and more neurons as we scale up the problem; generalization makes little sense in the context of the random problem. An entire spectrum of mappings falls between these limits. We can include any level of correlation between bits in the input strings. A priori knowledge of what level and type of correlations enter into the problem bears on the choice of network architecture and representation of the input data. The level of correlation in important classes of problems (e.g., vision) is not arbitrary [16], so we can always constrain our choices of architecture and representation.

Decomposing the input space of the Choice Problem into disjoint clusters reveals the types of correlation in the problem. For example, in the nonrandom Choice Problem, we define a Boolean function, which performs an *OR* operation on the set

$$A = \{x_i | i = 2n + 1, n = 1, 2, \ldots\}, \tag{4.1}$$

where x_i is the ith bit in an input string. Only points that return "true" belong to the inverse mapping of $(-)$. Clusters form arbitrarily in the random Choice Problem. No underlying rule or operation governs what clusters appear. Only the statistics of the clusters, which we determine from considering an ensemble of realizations, matters. Quantities like the probability of observing a cluster of size S (S is the number of points in the cluster) are the meaningful ones.

The analysis of the Choice Problem shows how generalization and learning in feed-forward networks sensitively depend on the geometric structure of the inverse mapping. A feed-forward network encodes mappings simply by partitioning the input space and the representation spaces of the hidden layers into disjoint regions: each linear-threshold neuron defines one hyperplane partition. Generalization occurs only when the training set provides enough constraints to position the hyperplanes properly. The back-propagation learning algorithm provides an automatic procedure to determine the appropriate partitions. Back-propagation, however, has severe limitations. Small changes in the definition of simple maps (such as from case A to case B) radically alter the computational complexity of the mapping and the learning performance of the network.

References

[1] John S. Denker (ed.), *Neural Networks for Computing* (American Institute of Physics, New York, 1986).

[2] John Denker et al., "Automatic Learning, Rule Extraction, and Generalization," *Complex Systems*, 1 (1987) 877–922.

[3] Stephen Jose Hanson and David J. Burr, "Knowledge Representation in Connectionist Networks," to be published (1987).

[4] Alan Lapedes and Robert Farber, "How Neural Nets Work," *Neural Information Processing Systems*, edited by Dana Z. Anderson (American Institute of Physics, New York, 1986) 442–456.

[5] P.M. Lewis and C.L. Coates, *Threshold Logic* (Wiley, New York, 1967).

[6] F. Rosenblatt, *Principles of Neurodynamics* (Spartan, New York, 1962).

[7] M.L. Minsky and S.A. Papert, *Perceptrons* (MIT Press, Cambridge, 1988).

[8] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation", in D.E. Rumelhart and J.L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations* (MIT Press, Cambridge, 1986) 318–362.

[9] James A. Anderson, Brown Univerity, private communication (1988).

[10] David C. Plaut, Steven J. Nowlan, and Geoffrey E. Hinton, "Experiments on Learning by Back Propagation," to be published (1986).

[11] Gerald Tesauro, "Scaling Relationships in Back-propagation Learning: Dependence on Training Set Size," *Complex Systems*, 1 (1987) 367–372.

[12] *Neural Network Workstation User's Guide*, Texas Instruments, (available from Michael T. Gately, Texas Instruments, M/S 154, Dallas, Texas 75265).

[13] Yaser S. Abu-Mostafa, "Neural Networks for Computing," in *Neural Networks for Computing*, edited by John S. Denker (American Institute of Physics, New York, 1986) 1–6.

[14] D. Stauffer, "Scaling Theory of Percolation Clusters," *Physics Reports*, 54 (1979) 1–74.

[15] B.B. Mandelbrot, *The Fractal Geometry of Nature*, (W.H. Freeman, San Francisco, 1982).

[16] David Marr, *Vision*, (W.H. Freeman, San Francisco, 1982).