Physica D 42 (1990) 307–321
North-Holland

GEOMETRIC LEARNING ALGORITHMS

Stephen M. OMOHUNDRO
International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA 94704, USA

Emergent computation in the form of geometric learning is central to the development of motor and perceptual systems in biological organisms and promises to have a similar impact on emerging technologies including robotics, vision, speech, and graphics. This paper examines some of the trade-offs involved in different implementation strategies, focusing on the tasks of learning discrete classifications and smooth nonlinear mappings. The trade-offs between local and global representations are discussed, a spectrum of distributed network implementations are examined, and an important source of computational inefficiency is identified. Efficient algorithms based on *k*-d trees and the Delaunay triangulation are presented and the relevance to biological networks is discussed. Finally, extensions of both the tasks and the implementations are given.

1. Introduction

Intelligent systems must deal with complex geometric relationships whenever they interact with their physical environment. The relationship between the motor signals sent to a set of muscles and the corresponding effect on the configuration of an organism's body in space is extremely complex and yet must be faithfully represented internally if the organism is to be able to effectively plan and carry out effective actions in the world. The relationship between the pattern of stimuli in different sensory modalities and the physical properties of the stimulus source is similarly geometrically complex. As if this intrinsic complexity was not enough, the perceptual and motor relationships change dramatically during an organism's lifetime as its size grows and its morphology matures. On an evolutionary time scale the change is even more dramatic and alternations to the organism's somatic form must be accompanied by altered internal models of its interaction with the world. A powerful strategy in the face of this variability is to base systems at least in part on learned instead of hardwired relationships. Aspects of the computations required must emerge during the interaction of the organism with the world.

As we attempt to build ever more complex and adaptive machines, we as engineers are faced with similar pressures toward systems in which at least part of the computational behavior is emergent. If successful, such systems should be far more robust and flexible than current hardwired systems. It is clear, however, that we are still at an early stage in the development of a discipline which will tell us which aspects of such systems can be made emergent and what the best approaches to implementation are. The three terms used in this paper's title suggest that such a discipline will be a symbiosis of the fields of mathematics (geometric), statistics (learning), and computer science (algorithms). While the joint interaction of these three fields together appears to just be beginning, there has been significant recent activity between them in pairs. Stochastic geometry studies statistical properties of geometric entities and much of classical statistics is being reformulated in a geometric coordinate-free form, the statistical analysis of probabilistic algorithms has become a central topic in theoretical computer science and computational issues have begun to be of critical importance to modern statistical analysis, and computational geometry and computational learning theory have flourished in the last decade. The time appears ripe for the development of a deeper understanding of emergent computation for the geometrical tasks essential to perception and motor control.

In this paper we will focus on the learning of relationships which have a geometric character to them. This is a small subclass of all possible emergent behavior, but it is one which is relevant to perceptual and motor tasks. We will examine a spectrum of implementation choices ranging from artificial neural networks to computational geometry algorithms and will identify some important trade-offs between them. We will see that in the network approaches a fundamental choice is the extent to which individual units are localized in their response. The use of localized units leads to faster and more reliable learning and is more robust in general, but global units may be more efficient if they are specifically tuned to the task at hand. We present a new biologically plausible implementation for a localized unit.

It is particularly clear in the systems with localized representations, but also true for those with global ones, that much of the computation involved in a typical retrieval is not actually used in determining the answer. We examine computational techniques for avoiding this computational inefficiency and describe some powerful data structures which have proven to be efficient, powerful, and easy to implement in practice. We describe a geometric construction known as the Delaunay triangulation and show that it is an optimal decomposition in a certain sense and that it may be efficiently implemented computationally. The algorithmic ideas presented apply to both serial and parallel implementations and the underlying concepts are generally useful for the construction of emergent systems. We describe how the basic concepts naturally lead to networks which employ strategies, such as focus of attention, which are commonly seen in animal brains. We also discuss extensions to both the emergent tasks and their implementations.

2. Two geometric learning tasks

Our motivating goal is the construction of the components of an intelligent system which interface with the physical world. On the input side this includes components for visual, auditory, and somatosensory perception, and on the output side, graphics, sound production and robotics. These components must provide the interface between the primarily geometric nature of the world and the apparently symbolic nature of higher intelligence. As Harnad [11] discusses, it is these interface systems which provide the conceptual grounding for internal symbols.

An example task of great importance is visual object recognition. It takes you perhaps half a second to visually recognize the volume in your hands as a book, yet no current engineered systems are capable of this task. Because this kind of processing happens so quickly and is mostly below the level of consciousness, people notoriously underestimate its difficulty. Artificial intelligence researchers in the early sixties consistently underestimated the time and computing power needed for intelligent computation. Partly motivating this optimism was a belief in a kind of "holy grail" of intelligence. The idea was that if one just found the right clever inference mechanism, a simple system could exhibit human level intelligence (apparently John McCarthy believed that a PDP-1 was a sufficiently powerful computer for human level performance [7].) Unfortunately, the early AI systems were found to be quite rigid and to lack common sense [3].

During the seventies interest shifted away from fancy inference mechanisms and toward the representation of task-specific knowledge. The success of a variety of expert systems showed that with enough knowledge even systems with quite limited inferential power could perform quite well. The phrase "knowledge is power" was used to describe the new emphasis. It is now believed that large amounts of knowledge about a domain are needed to achieve human level performance. Recent estimates suggest that people know at least 70 000 "chunks" of information in each of the domains in which they are expert [30]. Most AI systems are currently constructed entirely by hand and an enormous effort would be needed to give them anywhere near this amount of knowledge. It is perhaps this realization which has stimulated the great interest in machine learning, neural networks, and other systems with emergent computation during the past few years.

We would like machines to build up their knowledge bases through experience, as people do. Simply remembering past experiences is a conceptually trivial task. The key to effective learning is the ability to *generalize* previous experience to new situations. It is difficult to precisely identify the nature of desirable generalizations because of the philosophical obstacles in inductive logic and the philosophy of science as discussed by Churchland [5]. Any criterion which a procedure uses to prefer one generalization over another, such as Occam's razor, is called the *inductive bias* of the procedure. Many different inductive biases have been studied in purely symbolic domains [22].

Unlike general domains, there is a natural inductive bias in geometric domains such as those of interest here. We might call this bias the *principle of continuity*. Unless a system explicitly knows otherwise, it should assume that geometrically nearby perceptions correspond to nearby states of the world. A system faced with a problem can apply previous experiences which were geometrically near to it. We will call problems in which this inductive bias is applicable "geometric learning" problems. Learning tasks are generally separated into "supervised learning", in which a teacher provides example inputs with corresponding desired outputs, and "unsupervised learning", in which there is no teacher present and the nature of the task is more amorphous. There are a large number of important tasks for which the basic point made here is relevant, but we will focus on the two most natural supervised geometric learning tasks: classification learning and smooth nonlinear mapping learning.

For both of our tasks, we will assume that the input to the system may be represented as a point in an n-dimensional Euclidean feature space $\mathbb{R}^n$. The geometric structure of this space is meant to capture the geometry of the input in the sense that inputs which are close in Euclidean distance in the space should have a similar structure in the world. For classification tasks, the output of the system is one of a discrete set of classes. For mapping tasks, the output lies in another continuous space and varies smoothly with changes in the input.

A topical example of classification is *optical character recognition*. There are now several products available for scanning documents as bitmap images and converting them to machine-readable text. After the images of individual characters have been isolated, the essential recognition step is to classify them as characters. Most of the commercial products work by extracting about ten or twenty real-valued features from the image of the character, such as the ratio of its width to height, the density of its darkened pixels, or Fourier components of its image density projected onto the horizontal and vertical axes. The extracted feature vector is sent to a classifier whose job it is to choose the ASCII character which produced the image. A wide variety of fonts are used in modern publications and current systems attain robustness by using *learning*. A system is trained by presenting it with a sample document along with the identity of its characters. The system must adjust its classification procedure on the basis of this example input. Any classifier defines a partition of the input space with a region corresponding to each possible output (see fig. 1). An adaptive classifier must learn this partition from a set of labelled examples.

An example of a system for the evaluation of smooth nonlinear mappings which was studied in my laboratory is provided by MURPHY, a visually guided robot arm [21] (see fig. 2). The system

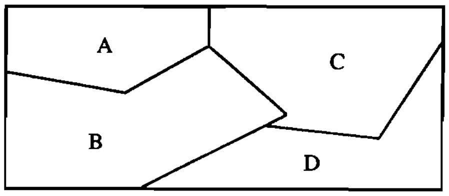

Fig. 1. The input space partition defined by a classifier.

controls three of the joints of robot arm and receives visual feedback from a video camera. Six white dots are painted on the arm and a real-time image processing system is used to identify their x and y coordinates in the image. The system thus has two descriptions of the state of the arm: a kinematic one represented by a vector in the three-dimensional space of joint angles, and a visual one represented by a vector in the twelve-dimensional space of image dot locations. Because many problems are specified in visual coordinates, but the system only has direct control over kinematic coordinates, one important task is for the system to predict the visual state corresponding to any given kinematic state. This is a smooth nonlinear mapping from $\mathbb{R}^3$ to $\mathbb{R}^{12}$. In ref. [21], this mapping was used as the basis for a variety of behaviors including reaching around visually defined obstacles to reach a visually defined goal.

The traditional approach to implementing a system of this type [27] would be to write down a set of model equations for the kinematics of the arm and for the imaging geometry of the camera, to form their composition analytically, and to evaluate the resulting mapping numerically. If the system is precisely constructed, such an approach can work well. With inexpensive or changing components, however, such an approach is likely to end up smashing the arm into the table. Biology must deal with the control of limbs which change in size and shape during an organism's lifetime. An important component of the biological solution is to build up the mapping between sensory domains by learning. A baby flails its arm about and sees the visual consequences its motor actions. Such an approach can be much more robust than the analytical one. MURPHY's camera had an auto-focus lens which caused the imaging geometry to vary dynamically as the arm moved. The system cheerfully succeeded in learning the effects of this complexity.

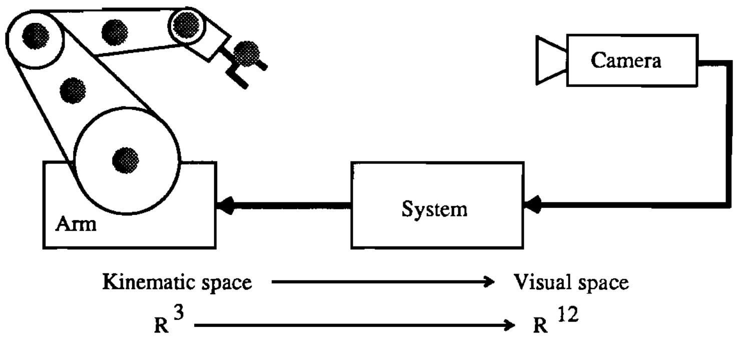

Fig. 2. The components of MURPHY, a visually guided robot arm.

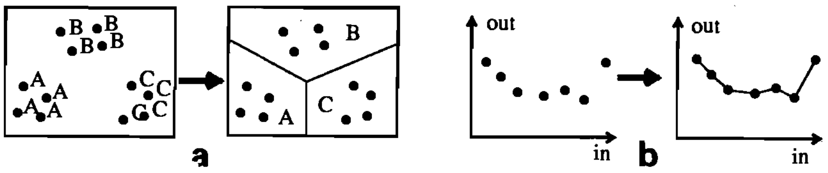

Fig. 3. (a) Classification learning. (b) Mapping learning.

We have identified two important geometric learning tasks. As shown in fig. 3, a classification learner must form a partition of the input space based on labelled examples. The geometric inductive bias should cause it to prefer to label nearby points similarly, leading to compact partition regions with small boundaries. Mapping learning is more like interpolation. We would like the learner to smoothly change the output as we vary the input. In ref. [25] we discuss a variety of other geometric learning tasks that arise in the visually guided robot arm domain.

3. Artificial neural network implementations

How might we build a system to perform these learning tasks? One currently popular approach is to use an artificial neural network with the backpropagation learning rule [20] (see fig. 4). In this approach the system consists of a number of model neurons connected in a feedforward network. To be specific, let us consider the robot arm task described above. The system is supposed to learn a smooth mapping from a three-dimensional space to a twelve-dimensional one. We may represent the input vector by the real-valued activities of three input neurons and the output vector by the real-valued activities of twelve output neurons. In addition there is an intermediate layer of units (usually called the *hidden layer*), each of which receives input from each input neuron and sends an output to each output neuron. The output of each model neuron is obtained by composing a linear combination of its inputs with a nonlinear "squashing function" such as a sigmoid. In forming the linear combination, each input to a neuron is multiplied by a "weight" which is meant to represent the strength of a biological synapse. For each setting of the weights the whole network computes a mapping from the input space to the output space. As we vary the weights, the mapping varies in a smooth but complex way. During learning the system is presented with inputs and the corresponding desired outputs. For any setting of the weights the mapping implemented by the network will make some errors compared to the desired mapping. If we estimate this error by taking the mean-squared error on the training set, we obtain an error function on the space of weight settings of the network. The backpropagation learning procedure just performs gradient descent on the error in this weight space. One cycles through the training set and on each example changes each weight proportionally to its effect on lowering the error. One may compute the error gradient using the chain rule and the information propagates backwards through the network, which accounts for the procedure's name.

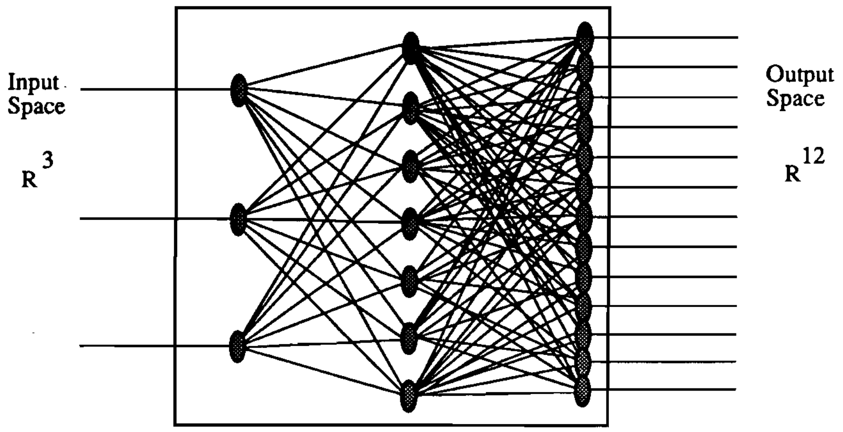

Fig. 4. An example backpropagation neural network for the robot arm learning task.

There has been much recent interest in this learning procedure partly because it is easy to implement and apply to a wide variety of problems. Unfortunately, a number of difficulties with it have become evident with experimentation. If one has an engineering task to solve, there are at present no techniques for obtaining bounds on how well a specific network will do on the task. The choice of how many hidden units to use is something of a black art. If there are too many, the network tends to just store the training examples and generalizes inappropriately. If there are too few, then it will not have sufficient representational power to approximate the desired mapping. The training procedure tends to be very slow and unpredictable, often depending strongly on the random starting conditions, and getting stuck at suboptimal local minima. Preliminary experiments indicate that the procedure does not scale well with the size of the problem [33]. If one is thinking of constructing large systems, it is a disadvantage that the activity of the units typically does not have a well defined "meaning" in terms of the input patterns. The procedure also appears to be biologically implausible.

Most of the problems with backpropagation in geometric domains stem from the fact that the units have global "receptive fields". By the receptive field of a unit we mean the region of the input space in which that unit has significant activity. The units in the class of networks we have described tend to have very large receptive fields. A single linear-threshold unit is active in the entire region of the input space which is bounded by a hyperplane. As the weights are varied, the hyperplane moves about. When the mapping is incorrect in a part of the space, it is the units whose receptive fields overlap that portion which have their weights adjusted. When receptive fields are global, most of the units must be adjusted for every kind of error. In correcting an error in one part of the space, there is a tendency to disrupt improvements made in another part. The large receptive fields are also largely responsible for the problems with local minima. Because a unit contributes to so many regions of the mapping, it is possible for it to get wedged in a position where any weight change is detrimental, even though there is a better state available. Several authors have discussed advantages of localized receptive fields [1, 35, 23].

The extreme of a network with localized receptive fields would have a separate input neuron for each small region of the input space, with no overlap. In fig. 5 we only show one output neuron. There will be as many such neurons as there are output dimensions. Learning a mapping in such a network is trivial. The partition of the input space shows the regions within which each input neuron fires. For any given input, only one input neuron is active. This makes the learning procedure extremely simple and fast. For each input/output pair, the system need only set the weight for the active input neuron at the value which produces the desired output level. This simple learning rule gives a piecewise constant approximation to the

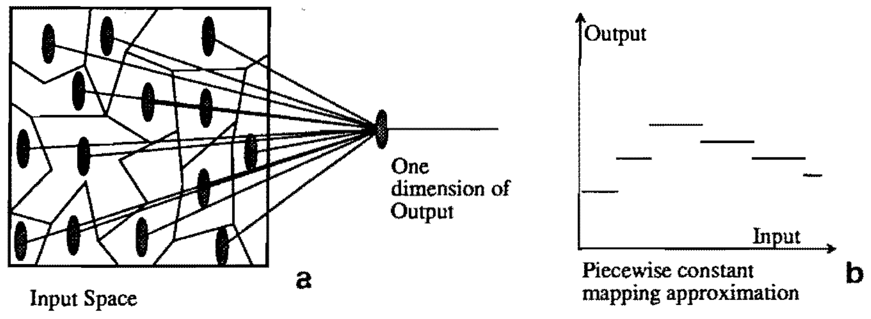

Fig. 5. (a) One of the output neurons in an extreme localist representation. (b) The resulting mapping approximation is piecewise constant.

mapping. It is easy to give explicit error bounds for a set of units representing a mapping with bounded Jacobian [24]. The receptive fields may also be adaptive to the underlying mapping, being larger in regions of small variation and smaller in regions of high variation. There are several approaches to adjusting the receptive fields automatically [15]. Such a system generalizes by assuming that the output should be constant over the disjoint receptive fields of the input units. For smooth functions, we can do much better than this. Consequently, the extreme localist representation tends to require a large number of units for a given level of accuracy.

We may take advantage of the smoothness of the desired mappings while still retaining many of the advantages of localization by allowing the input receptive fields to overlap somewhat. In fig. 6 we see the activities of six input neurons as a function of a one-dimensional input. The response is peaked at the center of each receptive field and dies down linearly away from it. Neighboring input neurons have overlapping receptive fields. We again assume a single linear output neuron for each output dimension which receives input from each input neuron. A simple learning procedure will allow the system to approach a piecewise linear approximation of the desired mapping. If each unit has the correct output when the input is at its center, then as we move from one center to the next, the output linearly interpolates between the two central values. Similar behavior may be achieved in higher dimensions by making a unit's receptive field be the star of a vertex in a triangulation and its response be linearly decreasing to zero in each bordering simplex. Again the receptive fields may be made adjustable, though now they need only be small in regions of high curvature rather than high variation. Again, precise error bounds may be obtained for mappings with bounded Hessian. The learning procedure is no longer one-shot, but must only simultaneously adjust the weights of units with overlapping receptive fields and so is fast, reliable, and analyzable. The degree of approximation and level of generalization of this kind of representation is far greater than with non-overlapping units, and yet most of the advantages of localization are retained. This kind of coding appears to be used extensively in biological nervous systems, especially near the periphery.

4. Why networks can be inefficient

We can see immediately the origin of computational inefficiency in networks with localized units. For each input presentation we must evaluate the state of each unit in the network, even though only a few of them contribute to the desired output. If you are only worried about the state of your hand, all of the computational capacity in the sensory system of your legs is wasted. This same kind of inefficiency occurs in the networks with global receptive fields, though it is less obvious. As an example, consider a perceptron classifier made up of linear threshold units. The hyperplanes corresponding to the input units partition the input space into regions. Within each region the classifier must produce the same response. The

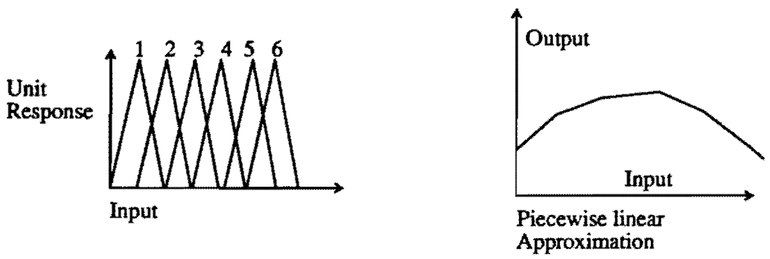

Fig. 6. The activities of six input neurons as a function of a one-dimensional input and the resulting piecewise linear approximation of the mapping.

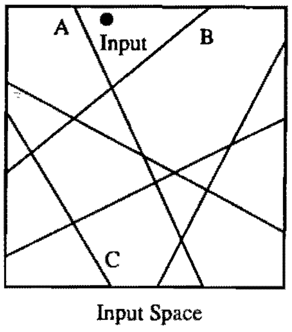

Fig. 7. A partition of the input space defined by perception hyperplanes.

perceptron learning algorithm adjusts the hyperplanes to fit the desired classification regions as well as possible. To determine the classification of an input we need only know which partition region the input lies in. A perceptron network determines this by computing the activity of each unit. This corresponds to testing the input against each hyperplane. Looking at fig. 7, however, we see that this task might be amenable to that basic trick of computer science: divide and conquer. Here the input space is two-dimensional and the hyperplanes corresponding to input units are just lines. Once we have determined that the input point is above lines A and B, there is no longer any need to compare it with line C; it must lie above it. In this manner we can use the results of earlier queries to rule out possibilities and avoid having to do computation. In section 5 we will discuss techniques for ruling out about half of the remaining possible questions for each question that is asked. In this way the number of questions whose answer must actually be computed is only the logarithm of the number of possible questions.

To see what effect such an approach might have on realistic computational problems, consider the robot arm task. Let us analyze the approach based on units with overlapping localized receptive fields. If we assign 50 units to represent each dimension, we need a total of $50 \times 50 \times 50$ or about 100000 input units. Each input unit synapses on each of the 12 output units, so there will be over 1000000 weights. On each input presentation, there is a multiplication and an addition associated with each weight, so there are at least two million floating point operations involved in each query. For any particular query, only about 50 of these operations are actually relevant to the output. Variants of the algorithmic approaches discussed in section 5 need only about 100 floating point operations to complete the task. The extra operations are used in the determination of which operations are actually relevant to the output. We see that even on this relatively small problem the speedup can be a factor of 20000. For larger tasks, speedups can be even more spectacular.

5. Algorithmic implementations

Let us now consider algorithmic approaches to the two problems under consideration. We will present specific solutions for illustration's sake, but there are many variations which may be superior in specific situations. The presentation is intended to emphasize algorithmic techniques whose basic principles are applicable to a wide variety of specific procedures.

5.1. Classification

There are many different approaches to learning a classifier from examples. In the asymptotic limit of a large number of examples, however, there is a universal classifier which does nearly as well as any other. Nearest neighbor classification simply assigns to a new input the class label of the nearest training example. A theorem proved by Cover and Hart [6] shows that if we describe the classes by probability distributions on the input space, then the probability of a classification error using nearest neighbor classification is asymptotically at most twice that of any other classifier. There are many variants on the basic scheme. For example, if there are labelling errors in the training set, it may be advantageous to let the nearest m neighbors vote on the class label.

While this basic procedure appears to depend on the notion of distance used, it is in practice quite robust. There are two length scales associated with a classification problem: the scale of significant variation in the underlying probability distributions and the typical spacing of training examples. Any of a wide variety of distance metrics which preserve the distinction between these two scales will give similar results. For a fixed number of samples, scaling one of the dimensions by a large enough number that the sample spacing becomes comparable to the scale of class variation along other dimensions will dramatically lower classification accuracy. For any fixed metric, however, such effects wash out as the number of samples increases. While there are many such practical issues to consider, we will simply consider an algorithmic technique for the basic task of finding the nearest neighbor.

Algorithms for finding nearest neighbors are studied in the field of *computational geometry*. This discipline was developed less than 15 years ago but has blossomed in recent years [10, 16]. Most of the work has been concerned with algorithms with good worst case performance. The most straightforward approach to nearest neighbor finding is to measure the distance from the test sample to each training sample and choose the closest one. This requires a time which is linear in the number of samples. Unfortunately, it does not appear that in high-dimensional spaces the worst case performance can be much better than this. If there are twice as many samples in a system, the retrieval will take twice as long. Most commercial speech recognition and optical character recognition systems on the market today use the straightforward approach and so are limited in the number of training samples they can deal with.

In practical applications one is usually concerned with good performance on average rather than with worst case performance, but theoretical analyses are then faced with the choice of distribution to average with respect to. The field of nonparametric statistics has developed techniques for analyzing the average properties of sets of samples drawn from *unknown* underlying distributions. Friedman et al. [13] used some of these techniques to develop a nearest neighbor finding algorithm which asymptotically runs in a time which on average is only logarithmic in the number of stored samples.

The algorithm relies on a data structure known as a *k-d tree* (short for k-dimensional tree). This is a binary tree with two pieces of information stored at each node: a dimension number d, and a value v. The nodes correspond to hyper-rectangular regions of the input space which we shall refer to as *boxes*. The root of the tree corresponds to the whole space. The two children of a node correspond to the two pieces of the parent's box that result when the dth dimension is cut at the value v. The left child corresponds to the portion of the box in which $x_d \leq v$ and the right to the portion in which $x_d > v$. As we descend the tree, the root box is whittled down to smaller and smaller boxes. The boxes of all the nodes at a given level in the tree form a partition of the input space. In particular, the leaf boxes form the *leaf partition*. Each box may be cut along any of the k dimensions and may be cut at any value which it includes. As a simple example, fig. 8 shows the partitioning of a three-dimensional input space ($k = 3$). There are four leaves in the tree whose corresponding box regions are labelled A, B, C, and D.

k-d trees are extremely simple and efficient to represent in the computer and yet they directly support a simple kind of geometric access. If we want to know in which leaf box a sample point lies, we need only descend the tree, at each node comparing the point's dth component with the value v. If it is less than or equal to v, we proceed to the left child, otherwise to the right, halting when we reach a leaf. The number of comparisons involved is equal to the depth of the tree. If the tree is balanced, it is logarithmic in the number of nodes.

For nearest neighbor finding, we need a k-d tree which is adapted to the training data. Each node is associated with the set of training samples contained in its box. The root is associated with the

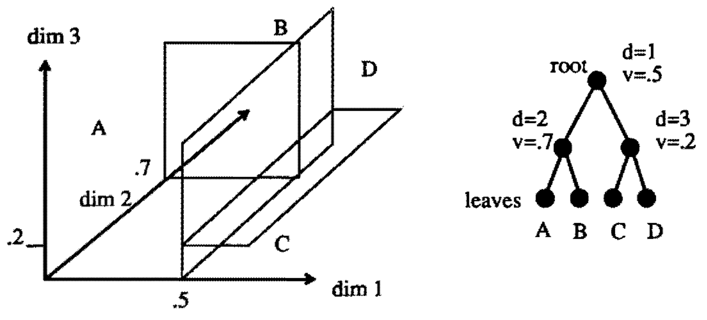

Fig. 8. The leaf partition of a three-dimensional input space and the corresponding k-d tree.

entire set of data. If we build the tree top down, for each node we need to decide which dimension to cut and where to cut it. There are several useful strategies here but the simplest is to cut the dimension in which the sample points associated with the node are most spread out and to cut it at the median value of those samples in that dimension. One might continue chopping until only one sample remains in each leaf box, but it is often useful in practice to leave a small number (e.g. 10) of samples in each leaf bucket. The expected shape of the leaf boxes under this procedure is asymptotically cubical because the long dimension is always cut. If the points are drawn from an underlying probability distribution, the expected probability contained in each leaf box is the same because cuts are made with an equal number of samples on each side. Thus the leaf partition is beautifully adapted to the underlying probability distribution. It chops the space into cubes which are big in the low-density regions and small in the high-density regions.

It is interesting to relate this strategy to Linsker's information theoretic analysis of neural adaptation strategies [18]. He suggests that a powerful way for networks to adapt is to adjust their properties in such a way that the mutual information between their inputs and outputs is maximized subject to any architectural constraints on the units. If the samples are subject to small additive noise, then the k-d cutting procedure maximizes the mutual information between an input sample and the "yes/no" question of which side of a cut it lies on. Using an argument similar to that in ref. [17] one can see that cutting at the median makes the output most informative and cutting the long dimension makes the volume of the cut as small as possible, minimizing the region of confusion caused by the noise. Of all the allowed k-d cuts at a given stage, the procedure chooses the one which is expected to give the most information about the input sample as we descend from the root. Successive "info max" cuts like this do not necessarily yield the "info max" tree but the resulting tree should be fairly close to it.

How is this adapted tree used to find the nearest neighbor of a test sample? As described above we find which leaf box the sample lies in log expected time. For many practical purposes, simply using the stored samples in this leaf box will be sufficient, yielding an extremely fast classifier. The nearest neighbor might not be in the same leaf box, however, if it is close to the edge of a neighboring box. To find it, Friedman et al. [13] present a branch and bound technique. We maintain the distance to the nearest sample point seen at any point in the algorithm. We descend the tree, pruning away any subtrees which cannot possibly contain the nearest point. If the entire box associated with a node is further from the test point than the nearest point seen so far, then the nearest neighbor cannot possibly lie within it. This branch of the tree need then be explored no further. Eventually all branches will have been pruned and the

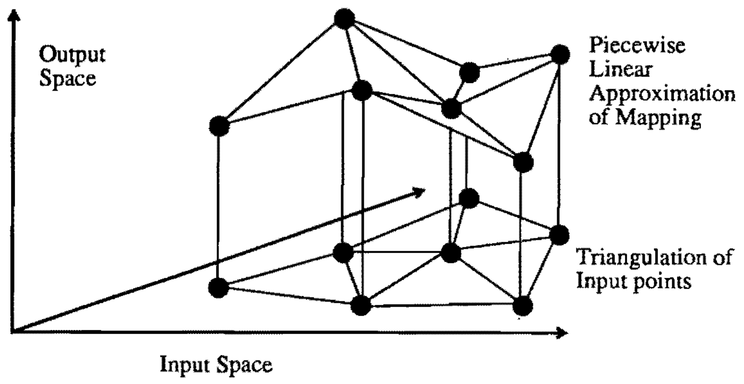
Output Space
Piecewise Linear Approximation of Mapping
Triangulation of Input points
Input Space

Fig. 9. An approximation of a mapping from a two-dimensional input space to a one-dimensional output space.

closest point seen so far is guaranteed to be the nearest neighbor. Both the nearest neighbor balls and the leaf boxes are big in low density regions and small in high density regions. Using a precise version of this observation, one may prove that on average only a constant number of leaves need be explored for any number of stored samples. The entire process is then asymptotically dominated by the initial logarithmic search.

5.2. *Mapping learning*

How might we apply similar ideas to the problem of learning nonlinear mappings? As we have discussed, we would like the system to interpolate between nearby training examples in evaluating the output for a test sample. Simple techniques, such as linearly interpolating between the values at the $k+1$ nearest neighbors can work well in certain circumstances [8, 12, 24], but leads to discontinuous approximations. A more well behaved approximating mapping may be constructed from any *triangulation* of the sample points. If the input space is k-dimensional, $k+1$ vertices are needed to define each primary simplex (i.e. higher-dimensional tetrahedron) in a triangulation. $k+1$ output values are also needed to perform linear interpolation. If we choose a triangulation, then linear interpolation of the vertex values within each simplex yields a continuous approximating function. Fig. 9 shows an approximation of a mapping from a two-dimensional input space to a one dimensional output space. There are two problems immediately suggested by this approach. The first is to find a criterion for selecting a good triangulation for approximation. In general, long skinny triangles will be bad, because the mapping may vary significantly in a nonlinear way along the long dimension. The second problem is to efficiently find the simplex in the chosen triangulation which contains the test sample, so that the linear interpolation may be performed. A nice solution to both of these problems may be had by using a special triangulation called the *Delaunay triangulation* [29].

The Delaunay triangulation is based on the fact that in a k-dimensional Euclidean space, $k+1$ points generically determine a sphere as well as a simplex. For example (see fig. 10) in two dimensions, three points determine both a circle and a

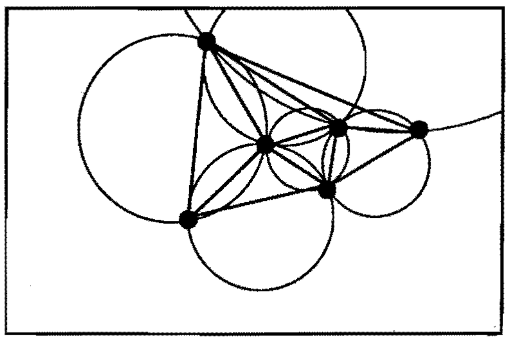

Fig. 10. The Delaunay triangulation of a set of points in two dimensions.

triangle. A set of $k+1$ sample points form the vertices of a simplex in the Delaunay triangulation if and only if the sphere which they determine does not contain any other sample points. The bounding spheres of the simplices in the triangulation are thus as small as possible and the triangles are as equilateral as possible. We have proven that among all mappings with a given bound on their second derivative, the piecewise-linear approximation based on the Delaunay triangulation has a smaller worst case error than that based on any other triangulation. The intuitive idea behind the result is quite simple. The worst error arises in approximating functions whose second derivative is everywhere equal to the maximum, i.e. with quadratic functions whose graph is a paraboloid of revolution and whose level sets are spheres. When we linearly interpolate through the values at the vertices of a simplex, the error will vanish on those vertices. The worst error function therefore has a level-set which is the sphere determined by the vertices of the simplex. The worst error occurs at the center of the sphere and is proportional to the square of the radius. At any input point the worst possible error is smaller when the spheres are smaller, so the Delaunay triangulation is best. It is straightforward to make this argument rigorously.

The Delaunay triangulation is also useful because the determination of which simplices are included may be made locally in a region of the input space. If we decompose the input space with a k-d tree built around the input samples, then the leaf boxes are small where the Delaunay spheres are small and large where they are large. A branch and bound algorithm very similar to the nearest neighbor finding algorithm may be used to determine the Delaunay simplex containing an input point in logarithmic time. Again we maintain the smallest Delaunay sphere whose simplex contains the point in question. We descend the tree, pruning away any branches whose box does not intersect the current Delaunay sphere and obtain the provably correct simplex in log expected time. It is not as easy to analytically obtain the probability distribution of Delaunay spheres as it is of nearest neighbor spheres, but even coarse bounds on the distribution of the radius yield the desired algorithmic performance.

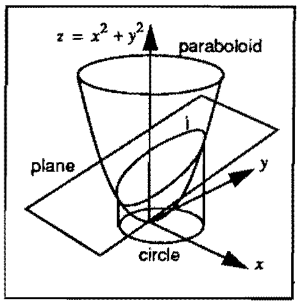

Fig. 11. The intersection of a plane and a paraboloid projects to a circle.

To compute the sphere determined by $k+1$ points, it is convenient to map the input space onto a paraboloid of revolution in a space of one higher dimension [29] by sending $(x_1, \ldots, x_k)$ to $(x_1, \ldots, x_k, x_1^2 + \ldots + x_k^2)$. Hyperplanes in this space intersect the paraboloid in the image of spheres under the mapping. To find the sphere determined by $k+1$ points one need only determine the hyperplane that their images determine. The Delaunay triangulation corresponds under this mapping with the convex hull of the images of the points. It is interesting to note that this construction shows that if a linear-threshold model neuron is given an extra input which is the sum of the squares of the other inputs, then its receptive field becomes a localized sphere. As the weights of the unit are varied, the center and radius of the sphere change. The exact form of the nonlinearity is not crucial, and in this way one may obtain biologically plausible neurons with localized receptive fields whose location and shape varies with simple linear weights. Fig. 11 shows a two-dimensional version of this construction.

6. Extensions

The basic ideas outlined here may be extended in a variety of ways to give algorithmic techniques for solving other important geometric learning tasks. Some of these are discussed in ref. [24]. Here we will briefly mention a few extensions that are under active investigation. The nonlinear mapping problems we have discussed have well defined input and output spaces. In many situations the system knows the values of different features at different times and from them would like to predict the values of the others. The relationship between the features may not be in the form of a mapping but instead may be a multi-sheeted surface or a probability distribution. The first case naturally leads to the task of *submanifold learning*. The system is given samples drawn from a constraint surface relating a set of variables. It must induce the dimension of the surface and approximate it. Typical queries might include predicting the values of unknown features when a subset of the features are specified (partial match queries). Geometrically this corresponds to finding the intersection of the surface with affine subspaces aligned with the axes. In situations with error, it should be able to find the closest point on a constraint surface to a given set. One approach to this problem is closely related to the Delaunay approach to mapping learning. If the surface to be learned is not self-intersecting and has bounded curvature (the analog of bounded Hessian), then we may prove that asymptotically a bounded radius variant of the Delaunay triangulation will converge on the surface with high probability. We only include spheres whose radius is less than the bound on the radius of curvature. In general, only simplices with some dimension less than $k + 1$ can be formed with this constraint and asymptotically this dimension gives the dimension of the surface. Branch and bound may be used to quickly find the closest point on the surface to a given point. This gives a nonlinear analog of linear pseudo-inverses.

k-d tree like structures are used in ref. [2] for a variety of statistical tasks. For many of these problems a generalization of k-d trees which we call *boxtrees* is superior. These structures store an entire box at each node and the boxes of siblings are allowed to intersect, yet they may be efficiently constructed and manipulated [26]. They are useful for learning and manipulating not only smooth constraint surfaces, but also smooth probability distributions using adaptive kernel estimation [9]. They support efficient random variate generation and Bayesian and maximum-likelihood inference. For distributions with fractal support they may be used to efficiently estimate the Hausdorff and information dimensions. For classification, variants which use information theoretic measure in construction are useful [14]. Boxtree structures also provide a powerful generalization of standard techniques for image decomposition [31].

These structures are also useful as components in larger systems. Graph structures which represent dependency relationships have become important in both probabilistic domains, where they are called influence diagrams or Bayesian networks [28, 32] and in deterministic domains, where they are called constraint networks [19]. Both of these domain types may be efficiently extended to continuous variables by using boxtree structures at the nodes. It appears that in this context algorithmic techniques may improve the speed and performance of the information propagation as well [4].

7. Relevance to parallel and network systems

While we have only described serial algorithms, many of the concepts are parallelizable. Omohundro [24] describes a massively parallel algorithm for k-d tree retrieval with almost optimal speedup on computers like the Connection Machine. When more exotic technologies are considered, algorithmic comparisons become more complex. While these algorithms eliminate much of the computation that is needed in straightforward network approaches, they still use as much memory to store the structures. In some technologies the cost of memory structures may be equivalent to the cost of computation structures, possibly nullifying some of the gains. In massively parallel systems, however, communication tends to be more of a bottleneck than computation. In this case, the wasted communication required by the straightforward network approaches will still cause systems which can prune away useless work to make better use of their hardware.

It is interesting to ask what relevance these considerations have for biological networks. Even though brains are large and highly interconnected, they are still extremely constrained by computational limitations. For example, a straightforward representation of the space of line segments on the retina might assign a segment unit to each pair of endpoints. Since there are about 10^6 fibers in the optic nerve, this would require 10^{12} units, more than using up our entire brain (this observation is due to Jerry Feldman). Similarly, it is easy to show that the completely connected networks favored by many backpropagation practitioners would lead to brains of enormous size if applied to any sizable chunks of cortex. Considerations such as these show that there are tremendous pressures on nervous systems for efficiency in representation, computation, and communication between areas. Neural hardware has properties which are very different from those of current computers and one expects the algorithmic trade-offs to be different. In particular, the locus of memory and computation appear to be the same in biological networks.

Some of the techniques that appear to be used in biological systems have the flavor of the algorithms described here. Each of the sensory modalities makes use of some form of focus of attention. Presumably this is a mechanism to devote higher-level hardware to only a portion of the data produced by lower-level systems. In this way a single piece of high-level hardware can be serially applied to different parts of the parallel sensory input. If this were not done, the high-level hardware would need to be replicated for each sensory focus. This kind of pruning of sensory input is very similar to the pruning of branches in the k-d tree to focus on only relevant knowledge. In general, hierarchical representations may be used to prune away possibilities at a coarse level and the strategy of proceeding from coarse to fine in the process of recognition is a common one. There appears to be psychological and neurophysiological evidence of some form of hierarchical representation in virtually every representation area of the brain.

The emergent aspect of the approaches described here arise from the construction algorithm which attempts to form a hierarchical representation which is optimally adapted to the system's experiences. One would like a general theory of such algorithms which would apply to both general algorithmic approaches and those bound by hardware constraints. The particular algorithms discussed here work in a top down manner and are off line (in that they need to be presented with all the data before they build their structures). Related structures described in ref. [26] may be constructed in a bottom up on-line fashion and are probably more similar in character to what is possible with dynamic networks. Understanding the formation of emergent hierarchical structure in realistic networks is a task of fundamental importance.

We have tried to give insights, in the context of specific examples, into some aspects of the nature of emergent computation in geometric domains. It is clear that a fundamental understanding of both the algorithmic and physical underpinnings of this kind of emergent computation will be critically important for both future science and technology.

Acknowledgements

I would like to thank Bill Baird, Kim Crutchfield, Herbert Edelsbrunner, Doyne Farmer, Jerry Feldman, Sally Floyd, Hans Guesgen, Ralph Linsker, Bartlett Mel, Nelson Morgan and Norm Packard for discussions of these ideas. This work was supported by the International Computer Science Institute.

References

[1] D.H. Ballard, Interpolation coding: a representation for numbers in neural models, Biological Cybernetics 57 (1987) 389–402.

[2] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, Classification and regression trees, Wadsworth International Group, Belmont, CA (1984).

[3] E. Charniak and D. McDermott, Introduction to Artificial Intelligence (Addison–Wesley, Reading, MA, 1985).

[4] P.B.-L. Chou, The theory and practice of bayesian image labeling, University of Rochester Department of Computer Science Technical Report No. 258 (1988).

[5] P.M. Churchland, Physica D 42 (1990) 281–292, these Proceedings.

[6] T.M. Cover and P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Information Theory IT-13 (1967) 21–27.

[7] J.D. Cowan, personal communication (1989).

[8] J.P. Crutchfield and B.S. McNamara, Equations of motion from a data series, Complex Systems 1 (1987).

[9] L. Devroye and L. Gyorfi, Nonparametric Density Estimation: The L1 View (Wiley, New York, 1985).

[10] H. Edelsbrunner and J. van Leewen, Multidimensional data structures and algorithms, A bibliography, IIG, Technische Universität Graz, Austria, Report 104 (1983).

[11] S.R. Harnad, Physica D 42 (1990) 335–346, these Proceedings.

[12] J.D. Farmer and J.S. Sidorowich, Predicting chaotic time series, Los Alamos National Laboratory Technical Report No. LA-UR-87-1502 (1987).

[13] J.H. Friedman, J.L. Bentley and R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Trans. Math. Software 3 (1977) 209–226.

[14] D.R. Hougen and S.M. Omohundro, Fast texture recognition using information trees, University of Illinois Department of Computer Science Technical Report No. UIUCDCS-R-88-1409 (1988).

[15] T. Kohonen, Self-Organization and Associative Memory (Springer, Berlin, 1984).

[16] D.T. Lee and F.P. Preparata, Computational geometry – A survey, IEEE Trans. Computers, C-33 (1984) 1072–1101.

[17] R. Linsker, Towards an organizing principle for a layered perceptual network, in: Neural Information Processing Systems – Natural and Synthetic, ed. D.Z. Anderson (American Institute of Physics, Denver, 1987) pp. 485–494.

[18] R. Linsker, Self-organization in a perceptual network, IEEE Computer (March 1988) 105–117.

[19] A.K. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1977) 99–118.

[20] J.L. McClelland and D.E. Rumelhart, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vols. 1, 2. Psychological and Biological Models (MIT Press, Cambridge, MA, 1986).

[21] B.W. Mel, MURPHY: A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning, University of Illinois Center for Complex Systems Research Technical Report No. CCSR-89-17A (1989).

[22] R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach, Vols. I, II (Kaufmann, Los Altos, CA, 1986).

[23] J. Moody and C. Darken, Learning with localized receptive fields, Research Report No. YALEU/DCS/RR-649 (1988).

[24] S.M. Omohundro, Efficient algorithms with neural network behavior, Complex Systems 1 (1987) 273–347.

[25] S.M. Omohundro, Foundations of geometric learning, University of Illinois Department of Computer Science Technical Report No. UIUCDCS-R-88-1408 (1988).

[26] S.M. Omohundro, Five balltree construction algorithms, International Computer Science Institute Technical Report, TR-89-063.

[27] R.P. Paul, Robot Manipulators, Mathematics, Programming and Control (MIT Press, Cambridge, MA, 1981).

[28] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (Kaufmann, San Mateo, CA, 1988).

[29] F.P. Preparata and M.I. Shamos, Computational Geometry, An Introduction (Springer, Berlin, 1985).

[30] R. Reddy, Foundations and grand challenges of artificial intelligence, AI Magazine (Winter, 1988) 9–21.

[31] H. Samet, The quadtree and related hierarchical data structures, ACM Computing Surv. 16 (1984) 187–260.

[32] R.D. Shachter, Evaluating influence diagrams, Operations Res. 34 (1986) 871.

[33] G. Tesauro, Scaling relationships in back-propagation learning: dependence on training set size, Complex Systems 1 (1987) 367–372.

[34] L.G. Valiant, A theory of the learnable, Commun. ACM 27 (1984) 1134–1142.

[35] D. Walters, Response mapping functions: classification and analysis of connectionist representations, in: Proceedings of the First IEEE Conference on Neural Networks Vol. III, San Diego, CA (1987) pp. 79–86.