Complex Systems **4** (1990) 543–572

A Genetic Learning Algorithm for the Analysis of Complex Data

Norman H. Packard*
Center for Complex Systems Research, Beckman Institute,
and
Physics Department, University of Illinois,
405 North Mathews Avenue, Urbana, IL 61801 USA

Abstract. A genetic learning algorithm modeled after biological evolution is presented to discern patterns relating one observable that is taken to be dependent on many others. The problem is reduced to an optimization procedure over a space of conditions on the independent variables. The optimization is performed by a genetic learning algorithm, using an information theoretic fitness function on conditional probability distributions, all derived from data that has a very sparse distribution over a very high-dimensional space. We will discuss applications in forecasting, management, weather, neuroanalysis, large-scale modeling, and other areas.

Introduction

We consider data obtained from a complex system, i.e., a system with many degrees of freedom that are highly coupled. Such data are usually beset with two problems: *(i)* randomness, from observational errors, from low-dimensional chaotic dynamics, and from high-dimensional noise, and *(ii)* limitations on the amount of data available. We will present a method for finding patterns in this data, especially applicable when no simple patterns are readily discernible.

We will assume the data to be a collection of pairs

$$\{(\vec{x}^{\,1}, y^1), \ldots, (\vec{x}^{\,N}, y^N)\}$$

where each $\vec{x} = (x_1, \ldots, x_n)$ is interpreted as a set of independent variables on which the corresponding y is presumed to be dependent. Observations of each $(\vec{x}, y)$ pair are obtained from a measurement that inevitably has a finite resolution, so values of these variables will take on one of a discrete set of values, $x_j^i \in \{1, \ldots, K_I\}$ and $y^i \in \{1, \ldots, K_D\}$. We will let K_I and K_D be

*Electronic mail address: n@complex.ccsr.uiuc.edu.

the number of states of the independent variable and the dependent variable, respectively.

For the moment, we need not attach any meaning to the label i of each datum. In some contexts, the data may come from observations of a single system at different times. In other contexts, the data may come from observation of many different systems (or subsystems) at a given time. Likewise, we will also refrain from making assumptions about the interpretations of the discrete values taken on by both the dependent and independent variables. They could be discretized versions of continuous variables, in which case they would inherit a natural order and metric, but the set of possible values may also represent labels for possibilities that have no particular connection to each other, as is often the case with "qualitative variables." No particular consistency is assumed for the data; there may be two samples, $(\vec{x}^{\,i}, y^i)$ and $(\vec{x}^{\,j}, y^j)$, for which $\vec{x}^{\,i} = \vec{x}^{\,j}$ but $y^i \neq y^j$. We consider only the case of one independent variable y, because this case is rich enough to capture the essential features of the method, and the generalization to high-dimensional dependent variables is fairly straightforward.

No assumption will be made about the distribution of points, although the methods we describe will be particularly useful when the distribution is broad and complex, i.e., when the $\vec{x}^{\,i}$ populate large regions of the space $\{1, \ldots, K_I\}^n$ with a distribution that is far from uniform. The goal of our analysis is to discern the nature of the average dependence of y on $\vec{x}$ by learning a pattern from the data. The basic tool to accomplish the task of learning will be a form of the genetic algorithm [1, 2].

One approach to searching for patterns in data is to try to construct a model that is "complete" in the sense that for any value of $\vec{x}$ given as an input, the model will give as an output its best guess for y. The present approach is different in the sense that the patterns it seeks to discover in the data are essentially incomplete. The discovered patterns will take the form of a set of hypotheses, each of the form, "*when some subset of the independent variables satisfy particular conditions, a certain behavior of the dependent variable is to be expected.*" It may happen that the model has no suggestions for a wide range of inputs because they do not satisfy the conditions of any of the discovered hypotheses.

Once the hypotheses are learned, they may be used in a variety of ways, depending on the context. They could be used to guide the adjustment of the system being observed, using the information of which variables are most relevant. They could be used for forecasting, in which case a query would be made by presenting a mystery point $\vec{x}$, and the expected behavior of y would be specified by the hypotheses satisfied by $\vec{x}$ (if any). They could be used for model building, where the hypotheses would tell which x_i were most important in determining y. We will discuss a range of applications after the method has been explained.

Logic and geometry

We will now discuss how statements about the independent variables having the form specified above may be identified with subsets of the space of independent variables. These subsets will be called conditional sets. The space of conditions explored is exhaustive if the set of all conditional sets is the same as the set of all possible subsets.

One easy way to formulate conditions for the hypotheses described above is to require particular coordinates in the space of independent variables to take on particular values. An example of such conditions is

$$C = \{x_{i_1} = c_1 \text{ AND } \ldots \text{ AND } x_{i_m} = c_m\}$$

where $c_1, \ldots, c_m$ are m constants. Such conditions specify a set of points,

$$X_C = \{\vec{x} \,|\, x_{i_1} = c_1, \ldots, x_{i_m} = c_m\}$$

For this particular set of conditions, X_C is a hyperplane in the space of independent variables, and the co-dimension of the hyperplane is the number of conditions m. If $m < n-1$, the set is sometimes called a "flat" instead of a hyperplane.

Setting particular coordinates to particular values implements the logical function AND on elementary propositions of the form $x_i = c$. We may also use an OR function to form conditions (and the corresponding conditional sets), for example,

$$C = \{(x_{i_1} = c_1 \text{ OR } x_{i_1} = c_1') \text{ AND } x_{i_2} = c_2 \text{ AND } \ldots \text{ AND } x_{i_k} = c_k\}$$

If we regard conditions for a particular coordinate, $\{x_i = c_1 \text{ OR } x_i = c_2 \text{ OR } \ldots\}$ as elementary propositions ξ, and represent the logical AND by the usual symbol for conjunction, $\wedge$, these conditions on k coordinates may be written compactly as

$$C = \xi_1 \wedge \xi_2 \wedge \ldots \wedge \xi_k \tag{1.1}$$

Geometrically the set $X_C \subseteq \{1, \ldots, K_I\}^n$ for such conditions is a set of rectangles in $\{1, \ldots, K_I\}^k$ crossed with $\{1, \ldots, K_I\}^{n-k}$. The rectangles all have their boundaries parallel to coordinate axes.

We may obtain more general subsets of $\{1, \ldots, K_I\}^n$ by taking the OR (disjunction, $\vee$) of conditions that have the form of equation (1.1),

$$\begin{aligned} C &= C_1 \vee C_2 \vee \ldots \vee C_p \\ &= (\xi_1^1 \wedge \cdots \wedge \xi_{k_1}^1) \vee (\xi_1^2 \wedge \cdots \wedge \xi_{k_2}^2) \vee \cdots \vee (\xi_1^p \wedge \cdots \wedge \xi_{k_p}^p) \end{aligned} \tag{1.2}$$

It is easy to see that the corresponding set X_C is related to $X_{C_1} \ldots X_{C_p}$ by

$$X_C = X_{C_1} \cup \cdots \cup X_{C_p}$$

Noting that

$$\text{NOT } (x_i = c) \Longleftrightarrow x_i = c_1 \text{ OR } x_i = c_2 \text{ OR } \ldots (\text{ for all } c_i \neq c)$$

because of the finite number of possible values for x_i, we see that the propositions $\neg\xi$ (NOT ξ) are contained within the set of possible propositions ξ as defined above. In this case, the conditions of the form (2) may be recognized as being in *disjunctive normal form*. An elementary theorem in logic [5] tells us that each possible Boolean expression involving the elementary propositions is tautologically equivalent to one in disjunctive normal form. Geometrically, this has the interpretation that all possible conditional sets X_C using conditions of the form (2) correspond to all possible subsets of the space of independent variables, $\{1,\ldots,K_I\}^n$.

Given a hypothesis specified by conditions C on the independent variables, we may now ask how well specified the y values are, given those conditions. This question is addressed by examining all the data points in the conditional set X_C. In particular, we must construct our empirical estimate of the conditional probability distribution of y values given that $\vec{x} \in X_C$,

$$P_C(y) = \frac{1}{N_C} \sum_{(\vec{x}\,y'),\in X_C} \delta(y - y')$$

where N_C is the number of points in X_C and the sum is taken over all points. $\delta(y-y')$ is 1 if $y = y'$ and 0 otherwise. This distribution may be collected from the data points directly.

We may now use $P_C(y)$ to evaluate the usefulness of the conditions in specifying y. This evaluation will allow us to assign a "value" or "fitness" (foreshadowing a biological analogy to be used below) of any conditional set X_C.

Intuitively, y is very determined by the conditions that specify C if $P_C(y)$ is very sharp. If y comes from a discrete measurement of a continuous variable, we may measure the sharpness of $P_C(y)$ simply by using the width, $\sigma = \sqrt{< y^2 > - < y >^2}$. The natural measure of sharpness, given σ, is $-\log\sigma$, which is proportional to the information contained in a sharp Gaussian of width σ.

Our expression for the fitness of a conditional set must also include a term related to the fact that $P_C(y)$ is only an estimation from a finite number of sample points. For example, if C only contains a single data point, then $P_C(y)$ is as sharp as possible, but may be spurious due to statistical sampling errors. We use a term proportional to $-1/N_C$ to "devalue" conditional sets with few points. Thus fitness of a conditional set may be expressed as

$$F(C) = -\log\sigma - \frac{\alpha}{N_C}$$

where α is a parameter to adjust the dependence on N_C, the number of points in h.

If y does not come from a continuous variable, the width σ cannot be used because it depends on a natural ordering and metric for y. In this case, the extent that y is determined by the conditions may be measured by the distance of $P_C(y)$ from the best possible *a priori* guess. In the absence of any knowledge, the *a priori* distribution is a flat distribution $\bar{P}(y)$, where

$\bar{P}(y) = 1/K_D$ for all y. An appropriate distance measure is the Kullback–Leibler function,

$$\begin{aligned} d(P_C, \bar{P}) &= \sum_y P_C(y) \log \frac{P_C(y)}{\bar{P}(y)} \\ &= H_{\max} - H(P_C) \end{aligned}$$

where $H(P_C) = -\sum P_C(y) \log P_C(y)$ is the entropy of the distribution $P_C(y)$, which has a maximum value of $H_{\max} = \log K_D$ when the $P_C(y)$ is flat.

Thus, we use as the fitness for the conditions C,

$$F(C) = H_{\max} - H(P_C) - \frac{\alpha}{N_C} \tag{1.3}$$

where we have again subtracted the term α/N_C to account for poor statistics.

It should be noted that this fitness function works perfectly well whether y is completely discrete or is derived from a continuous variable. Also, instead of using $\bar{P}$ as the *a priori* distribution, $P_0(y)$, the distribution of y values with no conditions at all, may be used. In this case, the distance function would not simplify as above, and the fitness would be

$$F(C) = \sum_y P_C(y) \log \frac{P_C(y)}{P_0(y)} - \frac{\alpha}{N_C}$$

The term $-\alpha/N_C$ is, at this point an *ad hoc* mechanism to adjust the dependence of the fitness on N_C. This term actually serves two purposes: (1) When the entropy-based fitness function is used, α may be chosen as to unbias the entropy estimate, where the bias comes from finite statistics [3]. (2) The term devalues the fitness of conditional sets that have small numbers of points, and hence have less statistical relevance. This second function served by the term is crucial, but this may not be the correct dependence on N to compensate for statistical relevance optimally. A more satisfying adjustment might use a calculation based on confidence intervals [4], but such calculations require an assumption of the distribution. The parameter α may be set with a recursive empirical procedure: start with an initial value of α, learn a set of good conditions, compute a confidence interval for each of the conditional distributions, and then if the statistics are too poor (reflected in a large confidence interval), raise α and begin again. Of course, the observer must eventually face the fact that a finite sample inevitably limits the confidence with which a parameter (like $H(P_C)$) may be estimated. Other fitness functions that measure statistical relevance more directly (using, for instance χ^2 measures) are being investigated.

So far we have emphasized the well-determinedness of y, measured by $-H(P_C)$, in the formulation of the fitness function. Other features may, however, be selected for, either in addition to or instead of, the determination of y. The fitness function is, in fact, the way for an observer's questions about the data to be encoded. One might, for instance, desire knowledge of what conditions give high average y values. One might also add a term

to enhance the fitness of simple logical conditions by adding a penalty term proportional to the number of coordinates with conditions, building in an "Occam's razor" as a part of the selection criterion. (Note, however, that there will be an Occam's razor effect even with the present fitness function, because an increased number of conditions will generally imply fewer points in the conditional set, which will consequently penalize the conditions' fitness.) In some contexts, one might even want to know what conditions lead to y being unspecified. Different possibilities will be discussed below in the section on applications.

Learning algorithm

The learning algorithm that is the crux of the method we describe is a version of the genetic algorithm [1, 2]. This type of algorithm takes its name from biology, where there is a straightforward analogy to the process of evolutionary learning. We will describe the algorithm first in general terms, then more specifically in terms of the present problem.

A genetic algorithm always has a population, each member of which is described with a set of genes $\vec{g} = (g_1, \ldots, g_n)$, where we will call each g_i a gene and the collection $\vec{g}$ a genome. Each member of the population may be assigned a fitness in some way, $F(\vec{g}) \in \mathrm{R}$. The nature and meaning of the genomes and of the fitness function over the space of genomes depends on the context. The fitness function typically encodes the problem to be solved. It must tell how to weight different combinations of genes in order to distinguish "good" combinations from "bad" combinations. It may be computed directly from the genes $\vec{g}$, or it may be obtained only indirectly, perhaps through the interaction of the genes with another system or systems. Despite the ambiguity in the interpretation of the genes and the fitness function, the genetic algorithm may be described independently.

The name "genetic algorithm" refers to the fact that the population of genomes $\{\vec{g}\}$ changes with time, and each generation is formed from the previous through a combination of a process of selection, using the fitness function, and a process of modification of the genes. The genes are modified in ways analogous to modification of biological genes; the maps that take old genes to new genes are called genetic operators, and we will consider the following types:

Mutation: $(g_1, \ldots, g_i, \ldots, g_n) \to (g_1, \ldots, g'_i, \ldots, g_n)$
with $g_i \neq g'_i$.

Crossover: to produce two new genomes from two old ones,

$$\begin{array}{l}(g_1, \ldots, g_i, g_{i+1}, \ldots, g_n) \\ (g'_1, \ldots, g'_i, g'_{i+1}, \ldots, g'_m)\end{array} \to \begin{array}{l}(g_1, \ldots, g_i, g'_{i+1}, \ldots, g'_m) \\ (g'_1, \ldots, g'_i, g_{i+1}, \ldots, g_n)\end{array}$$

The most general form of crossover does not depend on any ordering of the genes; rather, it takes a random subset of genes from one genome and exchanges them with the corresponding genes of another.

The dynamics of the genome may now be given by the following sequence of events:

0. Initialize the population, typically with a random set of genomes $\{\vec{g}^{\,i}\}$.
1. Calculate the fitness of all genomes using the fitness function.
2. Order the population by fitness.
3. Discard a fraction of the population with low fitness, and replace the deleted members with alterations of the remaining population, using the genetic operators.
4. A generation is completed. Go to step 1 and repeat.

Variants of the genetic algorithm perform step 3 in different ways, often using stochastic removal weighted by fitness.

This evolutionary procedure creates a dynamic on sets of genomes, and we will use the procedure to perform an optimization over the space of conditional sets. One of its advantages is that it searches as many hyperplanes as are kept in the population in parallel, analogous to the way nature performs a parallel search in the space of organisms.

Whether the genetic algorithm works, and how well, are questions that have no general answers, and instead depend very much on the details of the fitness function and its relationship to the genetic operators. The genetic operators define a metric on populations; one population is near another if every member of one may be obtained by few genetic operations on members of the other. This metric is called an operator-induced metric by Holland [1]. Point mutation alone corresponds to the metric of Hamming distance. Crossover makes changes that are very large with respect to the Hamming distance metric, and it corresponds to a metric of its own, giving a new distance function between sets of genes. The genetic algorithm will work well if the peaks of fitness function are continuously approachable with respect to the metric induced by the genetic operators used.

To use the genetic algorithm, we must identify the "genes" in our problem, identify a suitable measure of fitness of a given genome, and specify how genetic operators will operate on them. We will identify a gene vector with a specification of logical conditions on the x_i and use one of the two fitness functions discussed in the previous section.

We will have as many genes as there are independent coordinates, n, identifying each of them with one of the coordinates. Each of the genes will be allowed to take on either a value of $*$, indicating no condition is set for the corresponding coordinate, or a sequence of numbers $(c_1, \ldots, c_k)$ indicating OR'ed values for the corresponding coordinate. For example,

$$(*, (5, 9), *, *, 7, *, *) \sim X_C = \{\vec{x} | (x_2 = 5 \text{ OR } x_2 = 9) \text{ AND } x_5 = 7\}$$

Crossovers occur as described above, with some subset of the genes g_i switching places. There are several different types of point mutations we consider separately,

$$* \rightarrow c$$

$$c \rightarrow *$$

$$c \rightarrow c'$$

$$(c_1, \ldots, c_k) \rightarrow (c_1, \ldots, c_k, c_{k+1})$$

$$(c_1, \ldots, c_k) \rightarrow (c_1, \ldots, c_{k-1})$$

Note that this set of logical conditions have the form of equation (1.1), generating X_C that are rectangles, instead of the more general form of equation (2), which would generate arbitrary subsets. So far, we have found this to be adequate, since unions of good rectangles can be obtained anyway by their separate presence in the population. Application of the genetic algorithm to conditions having the form of equation (2) would require a similar symbolic representation with its own genetic operators.

One nonstandard feature of the genetic algorithm used here is a "diversity booster" mechanism. The way it works is that after ordering the population by fitness in step (2) above, using a fitness function such as equation (1.3), the fitness of each member of the population is reevaluated, attenuating the original fitness by a factor $0 \leq \beta \leq 1$ every time one of the g_i for a member matches a gene in the same location for any member more fit. This refitting procedure may be repeated and has the effect of devaluing members of the population that are near (with respect to Hamming distance) to other more fit members. Note that without this procedure, the fitness of the kth ranked member of the population will always increase monotonically with time, but with the procedure this is not true.

After running for some length of time, the genetic algorithm will find conditions that lead to fit conditional distributions. Evaluation of how useful the patterns actually are, is somewhat subtle, however. They may be fit, as measured by whatever fitness function is used on the space of conditions, but the question remains of whether the observed fitness is statistically significant. The genetic algorithm is mining the data, and is very effective at finding statistical flukes.

This question may be answered by comparing the fitness learned from the data with the distribution of fitnesses of conditions learned from a distribution of test data sets generated with a particular null hypothesis. For example, the null hypothesis is commonly no correlation between data samples, i.e., a distribution that is independent and identically distributed (IID). Particular realizations from this distribution are easily created by taking the original data and shuffling it well, destroying correlations while maintaining the original distribution. For each of many realizations, the learning

algorithm may be run, and a distribution of "fitness of the fittest condition learned" may be formed. The fitness learned from the original data must then be compared to this distribution. If it is near the mean (measured in units of standard deviation of the distribution), the learned pattern is not very significant. If it is far from the mean, it may be counted as significant. Unfortunately, this procedure is very time consuming. In the examples below, it is skipped, and only the results of the algorithm applied to the original data is reported.

In this particular application of the genetic algorithm, fitness is derived from the conditional probabilities obtained by "filtering" the data through the conditions specified by the gene vector, accumulating only those data points that satisfy the conditions to estimate the distribution $P_C(y)$.

In the parlance of machine learning, we might call the x_i coordinates of a high-dimensional *feature space* and y a *classification variable*. After many generations, the learning algorithm learns what features are relevant (i.e., those with non-$*$ genes), and in forming logical conditions on the relevant features, it is *inducing concepts* or *hypotheses* directly from the data [6–8].

The genetic operators provide creativity in the process of inducing new concepts. They act blindly, and their action is shaped by the process of selection in accordance with the fitness function.

Qualitative data

The first example we will consider is a case where the variables are qualitative, in the sense that they are not derived from continuous variables, but intrinsically discrete.

The data actually comes from the office of management and the budget of the *Regione Lombardia*, a state government in Italy, headquartered in Milano. The government has 730 offices to which it allocates a budget every year. Each office goes through a decision procedure to decide how to spend its money and at the end of the year has spent a certain fraction of it. The director would like to structure the decision procedures so that they lead to efficiency of the offices. To this end, every year he has all the offices fill out a questionnaire detailing their decision procedures by breaking them into nineteen procedural elements, each of which is represented by a variable that can take up to twelve different values. He would like to look at the data for the year to discern a pattern that could lead to advice for the offices to structure their decision procedures for maximal efficiency.

The data may be cast in an appropriate form quite easily. The independent variables $\vec{x}$ are the nineteen procedural elements, and the dependent variable y is the yearly efficiency (money spent divided by money allocated). In the 19-dimensional space there will be as many points as offices. The distribution of data points turns out to be far from uniform; in fact, more than 60% of the offices spend either all their money or none of it.

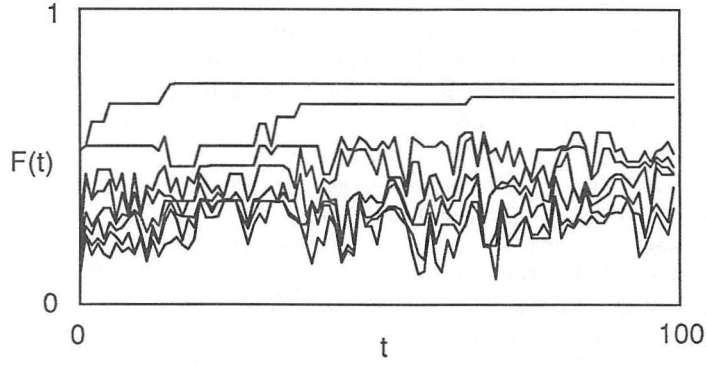

Figure 1: Fitness as a function of time for the eight fittest hyperplanes learning conditions on the decision procedures for money spent in many different offices of *la Regione Lombardia*. A population of 100 conditions was used, with 20 kept each generation.

The evolution of a population of hyperplanes is shown in figure 1, where the fitness of the top 8 members of the population is shown as a function of evolutionary time. The learned conditions were

*	*	4	*	*	6	*	*	*	*	*	2	*	*	*	*	*	*	*	4.129	19
*	*	*	*	*	*	*	*	*	26	*	*	2	*	*	*	*	*	*	3.976	12
*	*	4	*	*	*	*	*	*	*	*	23	*	*	*	*	4	*	*	3.305	16
*	*	*	*	*	*	*	23	*	*	*	*	*	*	19	48	*	2	*	3.142	4
*	*	23	2	2	*	*	1	*	*	2	*	*	*	*	*	*	*	*	3.064	14
4	*	*	*	*	*	34	*	*	*	*	*	2	*	*	*	*	*	*	3.052	8
*	*	*	4	*	*	*	*	*	26	*	*	2	*	*	*	*	*	*	2.898	12

where the occurrence of double digits implies the OR of two states for that variable and where the two columns on the right are the fitness and the number of points found in the conditional set. As the figure shows, the fittest members of the population are robustly so, and there remains a fair degree of competition in the rest of the population.

Management interprets the data by taking each of the learned conditions and looking at the resulting distributions of office spending efficiency (the dependent variable y). For the conditions to have been selected at all, the distributions must be well determined; the first step of the director is to see whether the average for the conditional distribution is high or low. If high, he will observe the corresponding conditions and consider having all offices use decisional procedural elements specified by the non-* entries in the

conditions. If low, he will consider having all offices use decisional procedural elements other than those specified by the non-* entries in the conditions.

The primary alternate method tried on these data was a type of cluster analysis, and the current method compared quite favorably.

Dynamics

A classical instance of trying to relate one variable to many others is found in the problem of forecasting. In this context, we will consider raw data in the form of a time series

$$\ldots, \vec{z}^{\,t}, \vec{z}^{\,t+1}, \vec{z}^{\,t+2}, \ldots$$

where $\vec{z}$ is a vector $(z_1, \ldots, z_n)$. If the dimensionality of the data is high enough and the dynamics are complex enough, then we might expect many or most of the coordinates of $\vec{z}^{\,t}$ to be irrelevant to the value of one particular coordinate, say z_0^t, in whose future we might be interested. The task of our data analysis in this case is to determine conditions on past values of $\vec{z}$ that are indeed relevant to future values of z_0.

The most obvious way to fit this task into the framework outlined above is to identify the value of z_0 at some time in the future, $z_0^{t+\tau}$, with the dependent variable, y, and a finite number of past states, $(\vec{z}^{\,t-k}, \ldots, \vec{z}^{\,t})$, with the independent variables, $\vec{x}$. The learning set is then as many temporally sampled $(\vec{x}\,, y)$ pairs as are available.

If the observable has only one dimension, using past states to form a higher-dimensional state–space representation of the dynamics is a common trick [10]. In this case, $z^{t-k}, \ldots, z^t$ would form k independent variables, and $z^{k+\tau}$ would be the dependent variable.

Example: Symbolic dynamics

Here, we use the reconstruction trick in a very simple example. We observe an orbit of the chaotic logistic map $x^{t+1} = f(x^t) = rx^t(1 - x^t)$ with r set to a value of 3.9, which generates chaos. Our observations are of a particularly simple form, $a^t = 1$ if $x^t > .5$ and $a^t = 0$ if $x^t \leq 0.5$. The independent variables were taken to be k consecutive symbols, and the dependent variable taken to be a symbol τ steps ahead of the last of the k symbols. Thus from raw data consisting of a string of symbols from a single orbit of the logistic equation with $r = 3.9$. The points for the data analysis were taken by moving a template along the data to obtain the training set.

$$\ldots 01\boxed{0010100101}011\boxed{0}010101111\ldots$$

The size of the left part of the template, and hence the dimension of the space of independent variables, in our examples is $k = 10$. Our first application is for predicting $\tau = 1$ steps into the future. This run used 10,000 data points.

For the first case, $\tau = 1$; the evolution of the eight fittest conditions is shown in figure 2. The eight fittest conditional sets of the population for a particular run, after 40 generations, were

* * * * * * * * 0 0	0.953	106
* * * * * * 0 * 0 0	0.739	57
0 * * * * * * * 0 0	0.596	55
1 * * * * * * * 0 0	0.479	51
* 0 * * * * * * 0 0	0.385	47
* 0 * * * * * 1 0 0	0.280	47
0 * * * 1 * * * 0 0	0.182	43
* * 0 * * * * * 0 0	0.162	39

The first number after the hyperplane specification is its fitness; the second number is the number of points in the hyperplane.

For a one-dimensional map, the degree of chaos is given by the Lyapunov exponent, which is equal to the average spreading rate of nearby trajectories. This quantity may be interpreted as the rate that initial information is lost, and as the rate that information is being produced by the dynamics. Under certain technical assumptions λ is equal to the entropy

$$\begin{aligned} h &= \limsup_{k\to\infty} \frac{H(k)}{k} \\ &= \limsup_{k\to\infty} \frac{-1}{k} \sum_{s^k} P(s^k) \log P(s^k) \end{aligned}$$

where $H(s^k)$ is the average information per symbol for blocks of size k. In fact, $\lambda \approx H(k)/k \approx H(k+1) - H(k)$, and has an observed value of about 0.718 in this case.

To make a connection between the fitness of our learned conditions and the information production of the system, we can re-express the entropy $h \sim$ [16]:

$$\begin{aligned} h &\approx H(k+1) - H(k) \\ &= H(s \mid s^k) \\ &= -\sum_{s^n} P(s^k) \sum_{s} P(s \mid s^k) \log P(s \mid s^k) \\ &= -\sum_{s^k} P(s^k) \sum_{s} P_{s^k}(s) \log P_{s^k}(s) \end{aligned}$$

where in the last line we have written the conditional probability to conform to our previous notation. The best learned conditional distribution is thus equivalent to a particular block distribution $P_{s^2}(s)$, where $s^2 = 00$.

From the final expression for the entropy, we see that the inner sum for a particular block s^k has the same form as the first term in the fitness

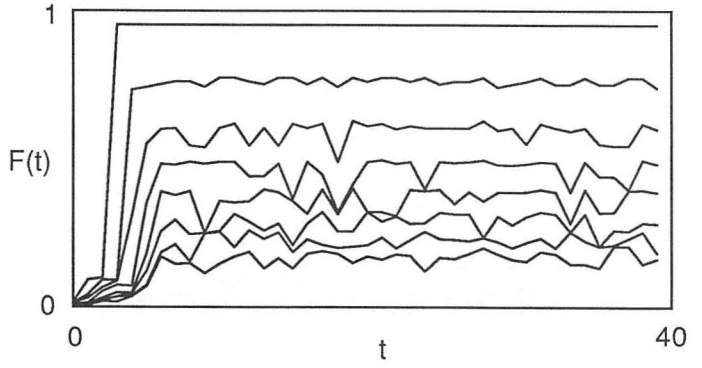

Figure 2: Fitness as a function of time for the eight fittest hyperplanes learning to predict the logistic equation for one time step into the future with one bit measurements. A population of 100 conditions was used, with 20 kept each generation.

function (1.3). The learning algorithm is learning combinations of sequences s^k whose conditional entropies make small contributions to the average over all s^k in the expression for the entropy h.

Probing the limits of predictability

For the logistic equation with $r = 3.9$, the Lyapunov exponent is $\lambda \approx 0.718$, so one bit degrades by that much on average every iteration [12]. Thus we expect some predictability one step into the future, but very little afterward. What we find, however, with the current method, is that there are indeed some predictable conditions that are learned. The eight fittest conditions discovered for predicting four time steps into the future after 25 generations were

* * 0 0 * 0 * 0 0 *	0.500	10
* * 0 0 1 * * 0 0 1	0.328	10
* * 0 0 1 0 * 0 0 1	0.157	10
* * 0 0 1 0 * 0 0 *	0.055	10
0 * 0 * * * * 0 0 *	0.040	35
* 1 0 * * 0 * 0 0 1	0.035	31
* 1 0 0 1 0 * 0 0 *	0.007	10
* 1 0 * * * 1 * 0 *	0.006	92

The reason for the existence of conditions that give such high predictability is that even though the average information loss is high, the local ex-

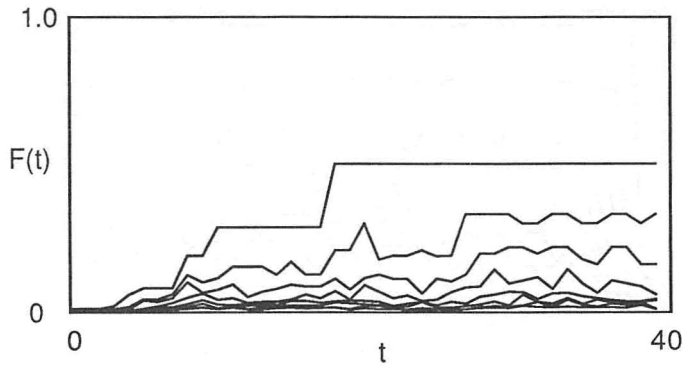

Figure 3: Fitness as a function of time for the eight fittest conditions learning to predict the logistic equation for four time steps into the future with one-bit measurements. A population of 100 conditions was used, with 20 kept each generation.

pansion on the attractor is nonuniform [13–15]. There are some pieces of the observed orbit of the logistic equation that pass over parts of the attractor where orbits are not spreading much, or where the orbits are even contracting. Thus these particular pieces of orbits are predictable. The conditions defining the conditional probability distribution may be mapped to subsets of the unit interval containing points whose trajectories satisfy the conditions, using straightforward techniques from symbolic dynamics [16]. Whether such conditions implying predictability may be found depends very much on the nature of the attractor being observed and is an aspect that is not captured in the numerical value of Lyapunov exponents (or metric entropy). For example, if we were observing symbol sequences from a piecewise linear "tent-map" with the same value of the Lyapunov exponent (i.e., exactly as chaotic), we would see no predictable conditions learned, because there are no parts of the attractor where orbits are convergent, as in the case of the logistic map.

This suggests the use of a new statistic for characterizing the predictability of a chaotic dynamical system. The usual measure of predictability, the Lyapunov exponent, for an iterated map of the unit interval, f, may be written [12]

$$\begin{aligned} \lambda &= \int P(x) \log |f'_x| \, dx \\ &= \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N} \log |f'|_{x^n} \end{aligned}$$

where

$$x^n = f^n(x^0), |f'|_x = \left|\frac{df}{dx}\right|_x$$

and the map f is assumed to have an asymptotic probability distribution $P(x)$ that is assumed to be ergodic and absolutely continuous with respect to Lesbesgue measure.

The discussion above suggests that rather than simply averaging $\log |f'|_x$ over the attractor, one should instead separate contributions to λ that come from orbit segments with different amounts of contraction along the orbit. To this end, we may define a finite time spreading rate for an orbit starting at a particular initial condition x:

$$\begin{aligned}\gamma_\tau(x) &= \log\left(|f'|_x |f'|_{f(x)} \cdots |f'|_{f^{\tau-1}(x)}\right) \\ &= \log |f'|_x + \sum_{i=1}^{\tau-1} \log |f'|_{f^i(x)}\end{aligned}$$

Following convention, we take the log base two, so that γ_τ has the units of bits.

The dynamics of f are predictable for time τ, with observations taken to have information I_0, if and only if there exist points on the attractor for which $\gamma_\tau(x) < I_0$. Such predictability will be useful to a real-world observer only if the set of such points has positive measure, so that $\gamma_\tau < I_0$ has non-zero probability of being observed, i.e., if it is possible to localize the systems to an initial state δ such that

$$\langle \gamma_\tau(x) \rangle_\delta = \int_\delta \gamma_\tau(x) P(x) dx < I_\delta$$

Following this idea, for any partition Δ, we can define the net predictability of the partition for time τ into the future to be

$$\Gamma_{\Delta,\tau} = \sum_{\delta \in \Delta} \theta(I_\delta - \langle \gamma_\tau(x) \rangle_\delta)$$

where we have used the function $\theta(x) = 0$ when $x < 0$ and 1 otherwise to give zero contribution to the terms in the sum that correspond to all the information in an initial condition I_δ being lost. We may then say that f is *predictable* for time τ given a measurement partition Δ if and only if $\Gamma_{\Delta,\tau} > 0$.

Much of the predictable character of a system is contained in the distribution of γ values for a given τ,

$$P_\tau(\gamma) = \int \delta(\gamma - \gamma_\tau(x)) P(x) dx$$

The predictability $\Gamma_{\Delta,\tau}$ is essentially summing over all parts of the attractor with $\gamma(x)$ small enough that $\langle \gamma_\tau(x) \rangle_\delta$ is small enough, so that we should have

$$\Gamma_{\Delta,\tau} \approx \int_0^{I_0} \gamma P_\tau(\gamma) d\gamma$$

for a suitable value of $I_0 \approx \langle I_\delta \rangle_\Delta$.

The characters of $P_\tau(\gamma)$ and $\Gamma_{\Delta,\tau}$ are clearly discernable for simple examples. Any map that has a critical point on the attractor that is not an isolated point will have $P_\tau(\gamma) > 0$ for γ arbitrarily small, and $\Gamma_{\Delta,\tau} > 0$ (i.e., the map is predictable) for arbitrarily large times τ. Naturally, for large τ, the measure of initial conditions that are predictable will be very small. On the other hand, a map that is uniformly expanding, e.g., the "tent map," has a $P_\tau(\gamma)$ that is a delta function, $P_\tau(\gamma) = \delta(\gamma - \tau\lambda)$, where the Lyapunov exponent, λ, is the log of the slope of the tent map. Thus, $\Gamma_{\Delta,\tau}$ is zero unless $I_0 > \tau\lambda$, and we recover the conventional wisdom regarding predictability [12, 17], since for this example local predictability is identical with long-term average predictability.

When the learning algorithm is presented with the task of learning to predict into the future by an amount τ, it learns, through conditions on the symbol sequences, sets of x that retain information τ steps into the future. The net amount of information that can possibly be learned, i.e., a bound for how well the learning algorithm can perform, is given by Γ_{τ,I_0}. Exploration of this and related measures of predictability will appear in future work.

These results also highlight an aspect of the present method that distinguishes it from other approaches to dynamical systems modeling. In other currently popular approaches, the state–space reconstruction is used to build a dynamical model that gives a good approximation to the evolution of orbits on an attractor that is presumed to be observed [18, 20, 22]. Here, the learning algorithm does not try to reproduce all orbits, but only to pick out selected pieces of orbits that are predictable. To make a prediction, one would present a query sequence to the learned hyperplanes, to see whether the query satisfies any conditions that make it predictable. For the latter case above, predicting four steps into the future, the best sequence was in fact fairly rare, happening only $\sim 1\%$ of the time. If the query point does not satisfy any of the conditions, no prediction could be made from the learned hyperplanes.

Continuous time dynamics

We will now consider how the analysis may be applied to more general contexts; here, the dynamics is an arbitrary continuous time signal. The data must again be reduced to a series of symbols (though not necessarily binary). This reduction is accomplished by sampling the signal and requires two choices: a sample interval, Δt, and a sample resolution, proportional to K, the number of possible values for a given sample of each continuous variable.

So in general, an n-dimensional vector $\vec{z}^{\,t}$ of continuous variables that change continuously with time, will be sampled to obtain a sequence of n-dimensional symbols $\vec{a}^{\,t} \in \{1, \ldots, K\}^n$:

$$\{\vec{z}^{\,t}\} \rightarrow \{\ldots, \vec{a}^{\,t}, \vec{a}^{\,t+\Delta t}, \vec{a}^{\,t+2\Delta t}, \ldots\}$$

The data for the learning algorithm would then be obtained by identifying the independent variables $\vec{x}$ with a collection of one or more sampled values, and the dependent variable with a value of a particular coordinate in the future, $y = a_i^{t+\tau}$.

The sampling choices are often made using the observer's intuition, for example, choosing Δt just small enough to capture temporal features that might be relevant. With the sampling resolution fixed, the learning algorithm will choose from the many possible past samples, those samples (intervals and sampled values) that specify the future to the greatest degree.

In this dynamical systems context, the learning of optimal samples is accomplishing the task of choosing coordinates that are most relevant to determining the system's future behavior. The x_i may be regarded as the coordinates of a high-dimensional reconstructed state space [10], and in this state space the learning algorithm performs a dimension reduction task, finding subsets of coordinates that are most important.

Our information theoretic fitness function over possible choices for coordinates is very much like the mutual information criterion for choosing an an optimal delay time [11]. It is also reminiscent of model criteria based on discrimination [22].

One difference in the present analysis is that in the computation of the entropy term in the fitness function, no average is taken over the independent variables, x_i; rather, the x_i are held at particular values in satisfying specified conditions. Another difference is that the learning algorithm yields a population of many good sets of conditions on the coordinates x_i. Geometrically, if we think of the observed dynamics in the reconstructed state space as being an attractor (possibly with added noise), each set of conditions corresponds to cuts through a projection of the attractor. If a set of conditions is fit, the corresponding cuts provide a "deterministic view" of a particular piece of the attractor. If the learning algorithm learns several distinct sets of conditions, it is telling us that different coordinates are relevant for obtaining the sharpest view of different deterministic pieces of the attractor.

Pyramids in time and space

So far, we have presented the sampling choices as a problem to be handled by the intuition of the observer. It would be nice, instead, to put these choices under the control of the learning algorithm. One method for accomplishing this end is to use the pyramid data structure and to add structure to the genetic algorithm.

The term "pyramid" comes from the original application in image processing. The idea of the data structure is to augment the raw data

successive stages of averages. For example, in the case of a single time-series z^t (which could be one of the coordinates of the more general $\vec{z}^t$ mentioned above), the pyramid is formed by

$$P_{11} = \frac{z^{t-n+1} + \cdots + z^t}{n}$$

$$P_{21} = \frac{z^{t-n+1} + \cdots + z^{t-n/2}}{n/2}, P_{22} = \frac{z^{t-n/2+1} + \cdots + z^t}{n/2}$$

$$\vdots$$

$$P_{b1} = z^{t-n+1}, P_{b2} = z^{t-n+2}, \ldots, P_{bn} = z^t$$

where P_{11}, the top of the pyramid, is an average over the past n samples; P_{2i} $(1 \leq i \leq 2)$, the second level of the pyramid, is two averages over $n/2$ samples; and so on, till the bottom of the pyramid P_{bi} ($b = \log_2 n$ and $1 \leq i \leq n$), which consists of the past n samples themselves. For convenience, we will always take n to be a power of 2, $n = 2^b$.

The pyramid values may be used directly as the independent variables, and a future value $z^{t+\tau}$ as the dependent variable, y, as before. The pyramid contains twice as many values as the raw samples, so the number of independent variables has doubled.

As before, there will be one "gene" for each independent variable:

$$\begin{array}{cccc} g_{11} & & & \\ g_{21} & g_{22} & & \\ g_{31} & g_{32} & g_{33} & g_{34} \\ \vdots & & & \\ g_{b1} & \cdots & g_{bn} & \end{array}$$

Since there are twice as many, the search problem is more difficult, but in return, we have the capability of detecting dependences on structures with different time scales more straightforwardly.

The genetic algorithm may also be structured to explore different temporal resolutions sequentially. In this case, only the top layers of the gene pyramid would be available for non-* initialization. Then, rather than allowing any * to mutate to non-* randomly, as before, we can allow finer

resolution to be explored only if there is a condition on the previous level. In other words, we allow

$$g_{ij} = * \longrightarrow g_{ij} = c$$

only if

$$g_{(i-1)j/2} \neq *$$

This allows us to oversample (i.e., to sample at very fine temporal resolution) and have a very large space of independent variables, but to keep the genetic algorithm from working too hard by ignoring the finely sampled variables unless they are particularly useful for identifying a feature.

If the data has inherent spatial structure as well as (or instead of) temporal structure, the pyramid technique may also be applied spatially. In this case, features with fine spatial detail would be selected only when relevant. A genetic learning approach to data analysis using a spatial pyramid data structure has already met with some success in the context of learning complex spatial dynamics [24].

Other state variables may also be formed using the pyramid technique. For example, instead of using simply the averaged values, differences between the averages on different levels may be used to form the Laplacian pyramid [23]. For the one-dimensional case, exploring different levels of the Laplacian pyramid is analogous to adjusting coefficients of a Taylor expansion.

The pyramid structured genetic algorithm is only one example of how structure may be added to the genetic algorithm in order to attack the basic problem of representation. The problem is that conventionally the genetic algorithm is given a fixed space of possibilities (i.e., gene configurations) to search, and this space depends on a particular representation for the possibilities. One would like, however, the choice of representation to be under the control of the learning algorithm as well, but this is problematic because the space of representations is generally much larger than the space of possibilities using a given representation. The pyramid structured genetic algorithm is one way to address the problem, by having a very large space of possibilities, including many possible representations, that is explored only gradually. Breaking through from one level of the pyramid being represented in the population to another is an extension from one representation to a larger one.

Similar representational extensions may be accomplished by exploring other functions of coordinates besides averages. For example, higher-order polynomials could be used, again explored gradually, with successively higher orders becoming available to genetic manipulation only after previous levels are activated. These tricks enlarge the space of representations available, but do not solve the problem, which is rooted in the choice of the original coordinates.

Other methods

The task of finding patterns from data, as we have formulated it, has been addressed by various "standard" statistical techniques, such as principal component analysis, factor analysis, projection pursuit, and cluster analysis. More recently, other different types of learning algorithms have also been used for similar types of analysis, especially in the realm of pattern recognition [8, 25, 26]. We are currently in the process of detailed comparisons, but a few general comments may be made even at this early stage.

The first, most general, comment is that this method should be particularly useful when the dimension of the space of independent variables X is very high, and when the patterns being searched for involve dependence of y on some small subset of the x_i. In this case, most previous methods waste much time considering the irrelevant coordinates, whereas the present method learns that are relevant and concentrates on them.

Previously used methods that are most similar in spirit to the present method are learning algorithms based on recursively cutting up a feature space (e.g., our $X = \{1, \cdots, K_I\}^n$), aiming to maximize classifiability of data points that lie in X. Here, the class of each data point would be our dependent variable y. The resulting partition of X is usually represented in a tree data structure sometimes called a "decision tree" or "kd-tree" [7–9,25,26]. Each node represents a hyper-rectangle, and daughter nodes represent the hyper-rectangles obtained by a cut. The top node of the tree is the whole space, and the tree is built by a divide-and-conquer algorithm. At each level, typically all possible cuts are examined and evaluated using some criterion. In fact, one of the most useful criteria is an information-theoretic one very much like our fitness, equation (3) [9].

The partition obtained from building the tree is analogous to the conditional sets $\{X_{C_1}, \cdots, X_{C_k}\}$ corresponding to the good learned conditions $\{C_1, \cdots, C_k\}$. One difference is that the conditional sets will typically *not* cover the entire space, whereas the tree-partition does. Thus the tree-partition attempts to give a global model rather than simply locating pieces of the space X_C that can be well modeled.

Another significant difference is that the tree partition is built by making a succession of incremental refinements, and "fitness" is a judgment on the refinements. In contrast, fitness in the present method is a feature of a particular partition element X_C that comes from simultaneously tested *global* conditions on the coordinates x_i. In some applications it has already been noted that an incremental approach can miss high-order informational correlations between coordinates [24].

One other difference is that the task of checking all possible cuts becomes very time consuming in the limit of a high-dimensional feature space. In this limit, maintaining the population can be less computational work. The current method will definitely be advantageous when the dimension of X is very high, and the dependent variable is dependent on only a small subset of the coordinates.

Another currently popular class of learning algorithms for pattern recognition problems is artificial neural networks [27]. The inputs to the network would be our independent variables x_i, and the output of the network would be a classification, i.e., the network's guess at a y value. Adjustment of the weights to obtain good classification corresponds to the genetic algorithm's search for good conditions.

Omohundro [25] has argued that the geometrical task implemented by neural network algorithms is quite analogous to cutting up the feature space as in the tree-partition methods discussed above. The problem is that they are not as efficient as the tree-partition methods on serial machines, because many of the nodes may be working needlessly. This general problem is exacerbated in the situation we address, that of a high-dimensional feature space with many of the features being irrelevant, typically. The neural network would spend needless time computing weights involving the irrelevant coordinates.

Statistical methods such as principal component analysis, factor analysis, projection pursuit, and linear discriminant analysis all use the general idea of trying to find good coordinates for X by a series of transformations of the original x_i. For this reason we will refer to them as "transformational methods." "Goodness" is usually measured by how well classification is accomplished by a hyperplane cut in the new coordinates.

Hyperplane cuts are well approximated by the conditional sets X_c in the present method, so it should be able to do at least as well as transformational methods. If the distribution is relatively simple, the present method may work harder than necessary. If the distribution is complex, the additional work will pay off in the discovery of richer structure than it is possible to see with the transformational methods.

Performing the transformations can also be very time consuming in a high-dimensional space. The genetic algorithm avoids much of this work by learning which coordinates are relevant and ignoring the rest. Once the "optimal" transformed coordinates are found, there are usually a few "most important coordinates," quite analogous to the collections of relevant coordinates that make up the learned conditions.

More detailed comparisons of this method with other methods will appear in future work.

Other applications

The method described above is designed to find patterns in data, where the patterns should address a question formulated by the observer. The application of the method involves two design issues: (1) putting the data in an appropriate form, and (2) encoding the question into the fitness function.

The form of the data we have used is that each data point consist of a pair $(\vec{x}, y)$, with all the x_i and y being integers. If the data is not immediately in this form, it must be preprocessed before the algorithm can be applied. For example, if data comes from "continuous" signals (i.e., signals with very

many states per sample), it must be thresholded and binned. A modified version of the algorithm that is better adapted to continuum variables (using inequalities for conditions) will be reported in future work.

We have so far concentrated on a particular question to be asked of the data, that of what logical conditions on the x_i lead to well determined distributions of y values, i.e., $P_C(y)$ that have low entropy. The desire to find such "good" conditions is directly translated into the fitness function of equation (1.3). However, as mentioned at the end of section 2, the fitness function may be altered to encode other desires.

Below, several example applications are listed, particularly with an eye toward how the two design criteria may be easily satisfied. In describing the applications, the main task will be specifying the form of the data; unless otherwise noted the fitness appropriate function will be that of equation (2). The solution of the criteria should not be considered unique; other ways of posing the data and questions may well exist. Also, the list of applications is by no means exhaustive. It is rather meant to be suggestive of many other possibilities.

Policy analysis

The section on qualitative data concerned the analysis of data that came from a government office, with the aim of using the analysis to make policy decisions. The development of the analysis for this context involved a fortuitous chain of events, but the example is by no means meant to be frivolous. It is, in fact, an example of a typical policy decision process that is complex in the sense that many elements must be considered in making the decision.

The first step in applying this method in policy analysis is obtaining the data. The existence of data for the example cited depended crucially on the intuition of the director of the office of management and the budget, G. Giorgi, that such data should be useful for his decision-making process. The application in other contexts would depend on similar initiative and commitment to gathering relevant data.

To obtain the data and put it into a computer, the decision process must be broken into many elements, with each element specified by a finite number of possibilities, which typically will correspond to different choices of action at a particular point in implementing the overall decision. The choices for each of the elements represents the specification of the independent variables x_i, so there are as many variables as elements to the decision process. Given a particular person's or group's decision process, there must be a way of evaluating the decision's performance (perhaps averaged over many decisions where the same process was used). This evaluation would be the dependent variable y. There will be as many data points as there are people or groups making decisions.

The independent variables may also be augmented with external conditions, such as weather conditions, etc. In fact, the learning algorithm's ability to learn which variables are relevant and which are irrelevant allows

one to test hypotheses about whether particular external conditions might be relevant to a decision's outcome.

The fitness function will usually be simply the usual information theoretic criterion. It could, however, be modified to give particular evaluation about the evaluation function for the decision procedure. For example, it could include a term proportional to the average evaluation for all the points in the conditional set X_C, in order to find conditions that led to high evaluations on average.

The example above, in the *Qualitative data* section, involved many offices within a government trying to spend money efficiently. Fiscal policy decisions play an important role, of course, but the method could be applied to almost any policy decision. In educational policy, for example, the evaluation function could be the average student performance, and the dependent variables could be all the decisions that determine curriculum, class size, integration level, teacher qualifications, etc.

Neural signals

The method is currently being applied to the analysis of neural signals. In this case the arena is that of evoked potential experiments. In these experiments, subjects are presented with different stimuli, for example A and B, and their electroencephalogram (EEG) is measured from scalp electrodes. From many sample signals (usually the order of hundreds), one would like to learn to classify a mystery input, to tell whether it was produced by stimulus A or B.

Here, the independent variables are the samples of the EEG are themselves the independent variables, and the dependent variable is the binary class variable A or B. One waveform is typically 260 samples, 5 milliseconds apart, so the dimension of the space of independent variables is 260. This is too high a dimension to search efficiently, so either the dimension must be reduced (e.g., by averaging over intervals), or the pyramid technique must be used. Both approaches are currently being tried; results will be reported in future work.

Medical diagnosis

Medical diagnosis is a classical pattern recognition problem, and the identification of the independent and dependent variables is quite straightforward. The independent variables are all the patient data, some of which will be entire time traces (e.g., how body temperature changes over a week). This data could also include patient information like sex, race, and other non-disease specific information that could, nevertheless, be relevant in finding a correlation with the disease. The dependent variable would be a discrete class variable indicating what disease actually occurred, as determined by pathology analysis, for particular set of symptoms (and personal information).

The list of possible diseases is large, and the set of possible symptoms is large, so large amounts of data are needed. Hopefully, the computer revolu-

tion will result in eventual automatic computer compilation of all symptoms, so that statistical tools like this one can achieve widespread use.

Visual pattern recognition

In visual pattern recognition, the usual formulation of the problem is to classify spatial patterns. The independent variables are obtained from a digitized image, and the dependent variable is the class to which the image belongs. The data should almost certainly be encoded in some way, such as the spatial pyramid discussed in the *Dynamics* section, with the corresponding structured genetic algorithm.

It may be that prior knowledge about the class of patterns being observed provides some information about what might be appropriate features for use as independent variables. For example, total power in different spatial frequency bands may be appropriate for classifying textures, and line-based features may be appropriate for optical character recognition. Additional features such as these could either substitute or supplement the original pyramid variables.

Weather

Weather analysis is probably the most prominent application in the wide realm of spatial dynamics. The problem can be seen as a special case of spatial pattern recognition discussed above. Weather data from satellites represents one of the largest data sets for any physical system, and this seems ripe for the application of these methods.

Global circulation models perform an essential role in weather prediction, but there are still many phenomena that lie outside its purview. One example is the sudden occurrence of mesoscale storm systems [30]. The learning approach presented here could provide a way to link preconditions with the occurrence of such storm systems.

The essential problem, as in all dynamical applications, is one of forecasting. One would like to take patterns of spatial data (temperature, etc.), and predict, for example, whether a mesoscale storm system will develop. The independent variables would be the space time data encoded in pyramid form. The dependent variable would be a binary classification variable denoting the occurrence or lack of a storm system.

In order to discern a pattern, there must be enough data to contain many examples of situations where storms both do and do not develop. $P_C(y)$ for this application will have only two entries, one for each value of y (storms or no storms). The uncertainty in the estimate of each of these probabilities is roughly $1/\sqrt{N}$, where N is the number of points in the probability histogram bin. Thus, to estimate the likelihood of a storm to within 10%, at least ~ 100 storm events must be available in the data.

Speech and language

Speech recognition is an area where learning algorithms have been applied with fair success; the current champion speech recognition system appears to be one based on a hidden Markov learning procedure [28].

The raw data for this problem is the digitized audio signal. The dependent variables could ultimately be considered to be actual words, though it is often conventional to break the recognition process into two parts, from signal to phonemes and phonemes to words. The phoneme-to-word problem has been successfully tackled by a neural-network-based learning scheme [29]. For the signal to phoneme application, the raw data in one-dimensional pyramid form (cf. the *Dynamics* section, above), would be appropriate as the independent variables, though these variables could be supplemented with other types of data derived from the raw samples, e.g., power-spectral components over finite windows. The dependent variable would be the phoneme present at a particular place in the wave form. One of the main problems for such applications is obtaining good learning data, since the identification of phonemes in natural speech is painstaking and often ambiguous. To reduce ambiguity, learning might be best applied to phoneme groups that have clear sonic boundaries.

The problem of learning language requires an even bigger leap between the data and the learned "dependent variables." Bigger than the leap from signal to phoneme, or signal to word, it requires the leap from signal to actions, from which an inference might be made from signal to mental state. The language problem involves not only the signal as raw data, but also the environmental context within which the signal was produced. Variables specifying environmental context could supplement the independent variables derived directly from the audio signal.

The most appropriate context for applying this method to the problem of language learning might be the problem of deciphering cetaceous speech. Dolphins, for instance, are known to have complex sonic interactions, and it has been hypothesized that some of the interactions might correspond to a linguistic interaction.

The problem of interspecies language learning immediately poses interesting problems in understanding the nature of language itself. One aspect of the problem is clear by realizing that many animals use sound for communication, e.g., a certain bird-call to indicate the presence of a predator, but one would probably not want to classify these as examples of language.

The bird-call does contain one element essential to language: symbolic representation. It does not, however, contain an essential richness of using linguistic symbols to build arbitrarily complex structures. The application of a learning-algorithm-based analysis could provide an empirical procedure to distinguish between linguistic and non-linguistic sonic interaction. For non-linguistic interactions, a relatively small set of simple patterns should be discovered, and for linguistic interactions, the discovered patterns should

be arbitrarily many, increasing as the space of independent and dependent variables is enlarged.

Large-scale modeling

For many large complex systems with many interacting nonlinear components, it is useful to construct large-scale models. The models may be used both for gaining a fundamental understanding of the mechanisms that generate particular features of the global dynamics, and also for practical purposes, to obtain a sense of what happens to the global dynamics when control parameters are changed. Such models have been constructed for ecological systems [31, 32] and economic systems [33].

Large-scale models all face a difficult problem: the system being modeled has very many degrees of freedom. To make the model as faithful as possible, model makers would like to include as many of these degrees of freedom as possible (some of the large-scale economic models have thousands of variables). On the other hand, the more variables in the model, the more difficult it is to decide on which variables should be included and how they should be coupled to the rest of the variables.

Thus, to build a model it would be useful to know which variables are most relevant to the changes of another. The analysis method outlined here could be used to determine which variables are relevant to which others, and hence what coupling schemes should be considered in the network of all variables.

To ascertain coupling schemes based on empirical relevance, the method should be applied in essentially the same way as outlined in the dynamics section, separately to every variable. Using the notation of the *Dynamics* section, $\vec{z}^t$ would represent the value of all the model variables at time t. For each of them, z_i, the analysis should be done separately, using ${z_i}^{t+\tau}$ as the dependent variable, and all the variables, $\vec{z}^t = (z_1^t, \cdots, z_n^t)$, as the independent variables. For each variable, z_i, the learned conditions would yield a list of independent variables (non-* entries in the genome) that are particularly relevant to z_i. In building the model, the equations of motion should then contain terms for the time dependence of z_i with explicit couplings to the learned relevant variables. For example, if z_{13} were found to have dependence on itself, z_{12}, z_{17}, and z_{23} (only), the equations of motion in the model should have an equation of the form

$$\vdots$$

$$\dot{z}_{13} = F_{13}(z_{12}, z_{13}, z_{17}, z_{23})$$

$$\vdots$$

More sophisticated modeling and map fitting techniques could possibly be used to help approximate F_{13} [25, 21].

Discussion

We have presented a method for discovering patterns in sparse, high-dimensional distributions using a genetic learning algorithm. The method is a nonparametric statistical method, in the sense that it makes no assumptions about the underlying distribution, and in the sense that it assumes no (parameterized) functional relationship between the dependent variable and the independent variables. The learned patterns take the form of conditions C on the independent variables x_i that lead to a well-determined distribution $P_C(y)$ for the dependent variable y.

As always in statistical analysis, one must be careful about inferring causality from the learned patterns, i.e., that y is caused by x_i satisfying conditions C. The patterns merely represent statistical correlations. In some contexts, the lack of causality is manifest; in the analysis of neural patterns, for instance, the dependent variable y represents a classification of the stimulus that caused the observed EEG signal. For the case of observing the dynamics, it is much more tempting to infer causality, since the dependent variable is taken to be in the future with respect to the dependent variables. In fact, this type of inference of causality is exactly what is used to build up networks of couplings between variables for the example application of large-scale modeling. It is possible, however, that the correlation is due to an indirect coupling; the method has no way of discerning this type of indirectness.

Inference of causality is just one example of the more general problem of extracting *meaning* from the learned patterns. The learning algorithm works blindly to find patterns, in the sense that it has no concern about what the variables actually represent. Meaning that an observer infers from the data inevitably involves interpreting the pattern based on the meaning of the variables (i.e., what quantities they actually represent). Nevertheless, the structure of the patterns represents a component of *intrinsic meaning* implicit in the observed data. The meaning extracted by the observer is a combination of this intrinsic meaning and interpretation-dependent meaning.

The structure discovered by the learning algorithm also gives one approach to quantifying complexity. The intrinsic complexity of the data may be equated with the number of conditions learned for the fittest logical conditions. This measure has the desirable properties of having a low value when the data are either completely random or very structured, and high for intermediate cases where randomness coexists with structure. Another example of measuring complexity using a learning algorithm is Crutchfield and Young's construction of ϵ-machines from symbolic data [34]. Both that method and the present one might be said to suffer from the fact that they are dependent on representation of the data, i.e., on the reduction from the raw data to symbols, and hence not a measure of intrinsic complexity of the data. It may be, however, that intrinsic complexity of data must, in principle, depend on representation, and that only if the representations used

by the learning algorithm are sufficiently rich, the learned complexity might approach a representation-independent bound.

Acknowledgments

I would like to acknowledge the stimulus and hospitality of G. Giorgi, *dirigente del l'Uficcio di Gestione, Regione Lombardia*, and B. Dente, who provided the data analysis problem that stimulated the original development of this method. An invitation by L. Galgani to lecture at the University of Milano was instrumental in stimulating the application to dynamics. For many subsequent ideas and help on writing the current version of PROPHET, the program that implements this method, I am indebted to T. Meyer. I have also had helpful conversations with A. Barron, M. Bedau, D. Farmer, L. Rendell, O. Rössler, and R. Shaw. The work has also been stimulated by many interactions at the Santa Fe Institute Economics Program. This work was supported by the Sloan Foundation, the National Science Foundation, Grant No. NSF Phy 86-58062 and the Office of Naval Research, Grant No. N00014-88-K-0293.

References

[1] J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (1975).

[2] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, MA, 1989).

[3] W. Li, "Mutual Information versus Correlation Functions," Center for Complex Systems Research Technical Report CCSR-89-1, University of Illinois (1989).

[4] Cramer *Mathematical Methods of Statistics* (Princeton University Press, 1946).

[5] H.B. Enderton, *A Mathematical Introduction to Logic* (Academic Press, New York, 1972).

[6] R.S. Michalski, "A theory and methodology of inductive learning," *Artificial Intelligence*, **20** (1983) 111; reprinted in *Machine Learning: An Artificial Intelligence Approach I*, R.S. Michalski, T.M. Mitchell, J.G. Carbonell, eds., (Tioga, 1983) p. 463.

[7] L. Rendell, "A general framework for induction and a study of selective induction," *Machine Learning*, **1** (1986) 177.

[8] J.R. Quinlan, "The effect of noise on concept learning," in *Machine Learning: An Artificial Intelligence Approach II*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell, eds. (Morgan Kaufmann, 1986).

[9] J.R. Quinlan, "Learning efficient classification procedures and their application to chess end games," in *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, T.M. Mitchell, J.G. Carbonell, eds. (Tioga, 1983) p. 463.

[10] N. Packard, J. Crutchfield, D. Farmer, and R. Shaw, "Geometry from a time series," *Phys. Rev. Lett.*, **45** (1980) 712.

[11] A. Frazer, "Using mutual information to estimate metric entropy," in *Dimensions and Entropies in Chaotic Systems*, G. Mayer-Kress, ed. (Springer, Berlin, 1985).

[12] R. Shaw, "Strange attractors, chaotic behavior, and information flow," *Z. Naturforschung*, **36a** (1981) 80.

[13] J. Nicolis, G. Mayer-Kress, and G. Haubs, *Z. Naturforsch*, **38a** (1983) p. 1157.

[14] R. Deissler and J.D. Farmer, "Deterministic noise amplifiers," Preprint, Center for Nonlinear Studies, Los Alamos, NM (1985).

[15] J.M. Nese, "Quantifying local predictability in phase space," *Physica D*, **35** (1989) p. 237.

[16] J.P. Crutchfield and N.H. Packard, "Symbolic dynamics of noisy chaos," *Physica*, **7D** (1983) 201.

[17] D. Ruelle, "Sensitive dependence on initial condition and turbulent behavior of dynamical systems," *Ann. NYAS*, **316** (1978) 408.

[18] A.J. Cremers and A. Hübler, "Construction of differential equations from experimental data," *Z. Naturforsch*, **42a** (1986) 797.

[19] P. Grassberger, "Information content and predictability of lumped and distributed dynamical systems," Technical Report WU-B-87-8, University of Wuppertal (1987).

[20] J.D. Farmer and J.J. Sidorowich, "Predicting chaotic time series," *Phys. Rev. Lett.*, **59** (1987) 845–848.

[21] J.D. Farmer and J.J. Sidorowich, "Exploiting chaos to predict the future and reduce noise," Preprint LA-UR-88-901 of the Theoretical Division and Center for Nonlinear Studies, Los Alamos National Laboratories (March 1988).

[22] J.P. Crutchfield and Bruce S. McNamara, "Equations of motion from a data series," *Complex Systems*, **3** (1987) 417–452.

[23] P.J. Burt, "Fast filter transforms for image processing," *Computer Graphics and Image Processing*, **16** (1981) 20.

[24] T.F. Meyer, F.C. Richards, and N.H. Packard, "A learning algorithm for the analysis of complex spatial data," *Phys. Rev. Lett.*, **63** (1989).

[25] S. Omohundro "Efficient algorithms with neural network behavior," *Complex Systems*, **1** (1987) 273.

[26] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees* (Wadsworth, Belmont, CA, 1984).

[27] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (MIT Press, 1986).

[28] K.-F. Lee, Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA (1988).

[29] T. Sejnowski and C.M. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, **1** (1987) 145.

[30] R.A. Maddox, D.M. Rodgers, and K.W. Howard, "Mesoscale convective complexes over the United States during 1981," *Monthly Weather Review*, **110** (1982) 1501.

[31] M. Conrad and H.H. Pattee, "Evolution experiments with an artificial ecosystem," *J. Theo. Bio.*, bf 28 (1970) 393; also, M. Conrad and M. Strizich, "Evolve II: A computer model of an evolving ecosystem," *BioSystems*, **17** (1985) 245.

[32] C.E. Taylor, D.R. Jefferson, S.R. Turner, and S.R. Goldman, "RAM: Artificial life for the exploration of complex biological systems," in *Artificial Life*, C. Langton, ed. (Addison-Wesley, 1989).

[33] S.A. Bremer, *Potentials of Globus and Related World Models.*

[34] J. Crutchfield, "Inferring statistical complexity," *Phys. Rev. Lett.*, **63** (1989) 105.