29

Machine Learning

Automated data-gathering techniques, together with inexpensive mass-memory storage apparatus, have allowed the acquisition and retention of prodigious amounts of data. Point-of-sale customer purchases, temperature and pressure readings (along with other weather data), news feeds, financial transactions of all sorts, Web pages, and Web interaction records are just a few of numerous examples. But the great volume of raw data calls for efficient "data-mining" techniques for classifying, quantifying, and extracting useful information. Machine learning methods are playing an increasingly important role in data analysis because they can deal with massive amounts of data. In fact, the more data the better.

Most machine learning methods construct hypotheses from data. So (to use a classic example), if a large set of data contains several instances of swans being white and no instances of swans being of other colors, then a machine learning algorithm might make the inference that "all swans are white." Such an inference is "inductive" rather than "deductive." Deductive inferences follow necessarily and logically from their premisses, whereas inductive ones are hypotheses, which are always subject to falsification by additional data. (There may still be an undiscovered island of black swans.) Still, inductive inferences, based on large amounts of data, are extremely useful. Indeed, science itself is based on inductive inferences.

Whereas before about 1980 machine learning (represented mainly by neural network methods) was regarded by some as on the fringes of AI, machine learning has lately become much more central in modern AI. I have already described one example, namely, the use of Bayesian networks that are automatically constructed from data. Other developments, beginning around the 1980s, made machine learning one of the most prominent branches of AI. I'll describe some of this work in this chapter.

29.1 Memory-Based Learning

The usual AI approach to dealing with large quantities of data is to reduce the amount of it in some way. For example, a neural network is able to represent what is important about a large amount of training data by the network's structure and weight values. Similarly, learning a Bayesian network from data condenses these data into the network's node structure and its conditional probability tables.

However, our growing abilities to store large amounts of data in rapid-access computer memories and to compute with these data has enabled techniques that store and use *all* of the data as they are needed – without any prior condensation whatsoever. That is, these techniques do not attempt to reduce the amount of data

before it is actually used for some task. All of the necessary reduction, for example to a decision, is performed at the time a decision must be made. I'll describe some of these *memory-based* learning methods next.

In Section 4.3, I mentioned "nearest-neighbor" methods for classifying a point in a multidimensional space. The "k-nearest-neighbor rule," for example, assigns a data point to the same category as that of the majority of the k stored data points that are closest to it. A similar technique can be used to associate a numerical value (or set of values) to a data point. For example, the average of the stored values associated with the k nearest neighbors can be assigned to the new point. This version of the rule can be used in control or estimation applications. The k-nearest-neighbor rule is a prototypical example of memory-based learning, and it evokes several questions about possible extensions.

First, to apply the nearest-neighbor rule (as I have presented it so far), each datum must be a list of numbers – a point or vector in a multidimensional space. So, one question is "How to represent the data so that something like the nearest-neighbor method can be applied?" Second, how is "distance" to be measured between data points? When the data are represented by points in a multidimensional space, ordinary Euclidean distance is the natural choice. Even in that case, however, it is usual to "scale" the dimensions so that undue weight is not given to those dimensions for which the data are more "spread out." If the data are not represented as points in a space, some other way of measuring data "closeness" has to be employed. Several methods have been proposed depending on the form of the data.

Third, among the k closest data points, should closer ones influence the outcome more than distant ones? The basic k-nearest-neighbor method can be extended by weighting the importance of data points in a manner depending on their closeness. Usually, something called a "kernel" is used that gives gradually diminishing weight to data points that are farther and farther away.

Fourth, what should be the value of k? How many nearby neighbors are we going to use in making our decision about a new piece of data? Well, with the right kind of kernel, *all* of the data points can be considered. The ones that are farthest away would simply have zero or negligible influence on the decision. The question about what value of k to use is now replaced by a question about how far away the influence of the kernel should extend.

Lastly, after all of the weighted neighbors are taken into account, how do we make a decision or assign a numerical value or values? Should it be the same as that associated with a majority vote of the neighbors or perhaps with some "average" of the weighted neighbors? Various versions of what are called statistical regression methods can be implemented depending on this choice.[1]

Andrew W. Moore (1965– ; Fig. 29.1) and Christopher G. Atkeson (1959– ; Fig. 29.1) are among the pioneers in the development of extensions to k-nearest-neighbor rules and the application of these extensions to several important problems in data mining and in robot control.

Experiments in applying these ideas to control problems are described in several papers. One paper[2] mentions the control of a robotic device for playing a juggling game called "devil-sticking." A memory-based system was developed to learn how to keep the stick in play. Figure 29.2 shows a schematic of a human doing the

Figure 29.1. Andrew Moore (left) and Chris Atkeson (right). (Photographs courtesy of Andrew Moore and of Christopher Atkeson.)

juggling. The robotic setup is also shown with some of the sensory and control parameters.

Besides applications in robotics and control, memory-based learning methods have also been used in other areas including data mining and natural language processing.[3]

29.2 Case-Based Reasoning

A subfield of AI, called "Case-Based Reasoning" (CBR), can be viewed as a generalized kind of memory-based learning. In CBR a stored library of "cases" is used to help in the analysis, interpretation, and solution of new cases. In medicine, for example, the diagnostic and therapeutic records for patients constitute a library of cases; when a new case is presented, similar cases can be retrieved from the library to help guide diagnosis and therapy. In law, previous legal precedents are used in interpretations of and decisions about new cases (following the legal practice of *stare decisis*, which mandates that cases are to be decided based on the precedents set by previous cases).

Cases that are similar to a new case can be thought of as its "neighbors" in a generalized "space" of cases. To retrieve close neighbors, the idea of closeness in this space must be based on some measure of similarity. One of the pioneers in case-based reasoning, Janet Kolodner (1954– ; Fig. 29.3), a professor of computing and cognitive science at the Georgia Institute of Technology, describes the process as follows:[4]

> Good cases [for retrieval] are those that have the potential to make relevant predictions about the new case. Retrieval is done by using features of the new case as indexes into the case

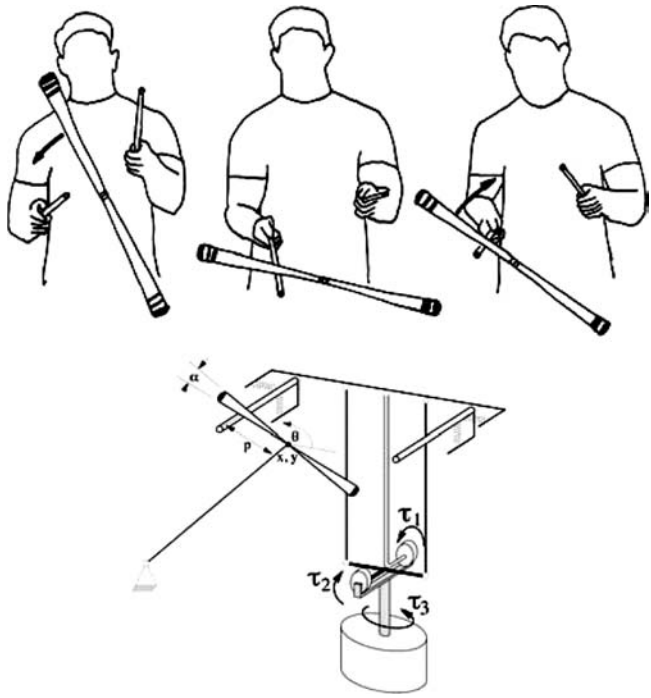

Figure 29.2. "Devil Stick" as played by a human and by a robotic memory-based learning system. (Illustrations from Christopher G. Atkeson, Andrew W. Moore, and Stefan Schall, "Locally Weighted Learning for Control," *Artificial Intelligence Review*, Vol. 11, pp. 75–113, 1997. Available online at http://www.cs.cmu.edu/~cga/papers/air1.ps.gz.)

library. Cases labeled by subsets of those features or by features that can be derived from those features are recalled.

[We then select from among these] the most promising case or cases to reason with.... Sometimes it is appropriate to choose one best case; sometimes a small set is needed.

When the retrieved case (or cases) is adapted to apply to a new case it might then (if it is successful) be revised so that the parts that might be useful for future problem solving can be retained in the ever-growing case library.

Case-based reasoning has roots in Roger Schank's model of dynamic memory (see p. 158). Early work was done by two of Schank's Ph.D. students, Janet Kolodner and Michael Lebowitz.[5] Another important source of ideas for CBR comes from Minsky's ideas about frames. Edwina Rissland (1947– ; Fig. 29.3), a professor at the University of Massachusetts at Amherst and another pioneer in CBR, writes[6] that her CBR work is a direct outgrowth of her "work on 'constrained example generation,' . . . which modeled the construction of new (counter) examples by modification of existing past 'close' examples (represented as frames) retrieved from a network of examples."[7] Rissland and her students have made important contributions to the use of CBR in the law.[8] She wrote me that the CBR process is sometimes summarized by the four "R's," Retrieve, Reuse, Revise, and Retain.[9]

Figure 29.3. Janet Kolodner (left) and Edwina Rissland (right). (Photographs courtesy of Janet Kolodner and of Edwina Rissland.)

According to a Web page maintained by the Artificial Intelligence Applications Institute at the University of Edinburgh, "Case-based Reasoning is one of the most successful applied AI technologies of recent years. Commercial and industrial applications can be developed rapidly, and existing corporate databases can be used as knowledge sources. Helpdesks and diagnostic systems are the most common applications."[10]

29.3 Decision Trees

Next on my list of new developments in machine learning is the automatic construction of structures called "decision trees" from large databases. Decision trees consist of sequences of tests for determining a category or a numerical value to assign to a data record. Decision trees are particularly well suited for use with non-numeric as well as numeric data. For example, a personnel database might include information about an employee's department, say, marketing, manufacturing, or accounting. In database parlance, data items like these are called "categorical" (to distinguish them from numerical data). In this section, I'll describe these structures, learning methods for automatically constructing them, and some of their applications.

29.3.1 *Data Mining and Decision Trees*

Data mining is the process of extracting useful information from large databases. For example, consider a database about peoples' credit card behavior. It might include payment records, average purchase amounts, late fee charges, average balances, and so on. Appropriate data-mining methods might reveal, among other things, that people with high late fee charges, high average purchases, and other identified features tended to have high average balances.

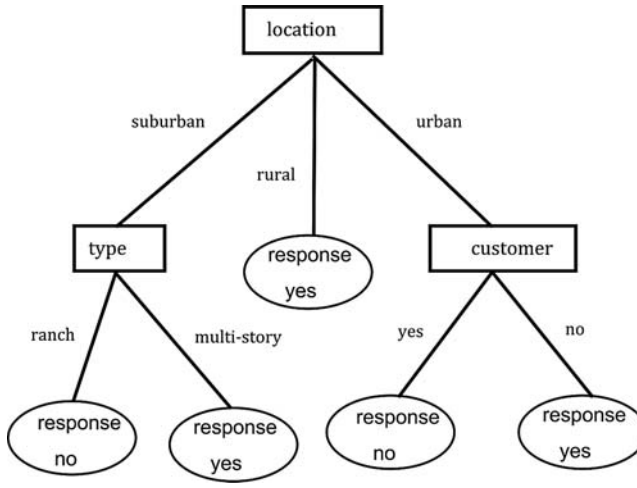

Figure 29.4. A decision tree for predicting responses.

One important data-mining method uses data to construct decision trees. Let's consider a very simple database to illustrate how decision trees work. Suppose a company, say Wal-Mart, maintains a database in which it stores information about households to which it had previously mailed discount coupons for some of its products. Suppose the database has information about the location of the household (urban, suburban, or rural), the type of house (either ranch or multistory), whether or not the household is a previous Wal-Mart customer, and whether or not the household responded to any of its previous coupon mailings. (Obviously, this is just a made-up illustrative example; I don't actually know anything about Wal-Mart's real databases.)

A tabular representation of such a database would look like this:

Household	Location	Type	Customer	Response
3014	suburban	ranch	yes	no
3015	rural	multistory	no	yes
...	...	...	...	...
5489	urban	ranch	yes	no

Each row in the table is called a "record." The items at the top of each column are called "attributes," and the items in a column are called the "values" of the corresponding attribute.

Analysis of this database, by methods I'll be explaining later, might reveal that the decision tree shown in Fig. 29.4 captured information about which households responded to the coupon mailing and which did not. The tests on attribute values are at the interior nodes of the tree (in boxes), and the results (whether or not there was a response) are at the tips (or leaves) of the tree (in ovals). Such a tree might be useful for making predictions about expected responses prior to sending out another mailing.

Methods have been developed to construct (that is, learn) decision trees like this one (and much larger ones too) automatically from large databases. I'll describe some of the history and how the major methods work.

29.3.2 *Constructing Decision Trees*

A. EPAM

Probably the earliest system for constructing decision trees was developed in the late 1950s by Edward Feigenbaum as part of his Ph.D. dissertation under Herbert Simon at Carnegie Mellon University (then called Carnegie Institute of Technology).[11] His system was called EPAM, an acronym for Elementary Perceiver and Memorizer. The goal of the research was to "explain and predict the phenomenon of [human] verbal learning." A standard psychological experiment for testing this ability involved showing people pairs of nonsense syllables, such as DAX-JIR and PIB-JUX. The first member of a pair was called a "stimulus" and the second a "response." After seeing a number of such pairs repeatedly, the subject is then shown a random stimulus and tested on his or her ability to generate the correct response.

Pairs like these were shown to EPAM during its "learning phase." Learning consisted in growing what Feigenbaum called a "discrimination net" for storing associations between stimuli and responses. The net was what we would now call a decision tree with tests on features of the letters at the internal nodes and responses stored at the tips or leaves of the tree. In EPAM's "testing phase," a nonsense stimulus syllable was filtered through the tests down the tree until a leaf was reached where (one hopes) the correct response was stored. A sample EPAM discrimination net is shown in Fig. 29.5. The round nodes are tests, and the boxed nodes are responses.

Not only did EPAM successfully model the performance of humans in this "paired-associate" learning task, it also modeled forgetting. Feigenbaum claimed that "As far as we know, [EPAM] is the first concrete demonstration of this type of forgetting in a learning machine."[12] EPAM was written in Carnegie's list-processing language, IPL-V. In fact, the list-processing features of languages such as IPL-V were required to write programs that could *grow* decision trees. Thus, it is not surprising that EPAM was the first such program.

Feigenbaum's program is still regarded as a major contribution both to theories of human intelligence and to AI research. Simon, Feigenbaum, and others continued work on EPAM programs, culminating in EPAM-VI, coded in IPL-V and running on a PC.[13]

B. CLS

The next significant work on learning decision trees was done at Yale University around 1960. There, psychologist Carl I. Hovland and his Ph.D. student Earl B. (Buz) Hunt developed a computer model of human concept learning.[14] After Hovland succumbed to cancer in 1961, Hunt continued work on concept learning and collaborated with Janet Marin and Philip Stone in developing a series of decision-tree learning programs called CLS, an acronym for Concept Learning System.[15] Hunt and his colleagues acknowledged the related prior work on EPAM.

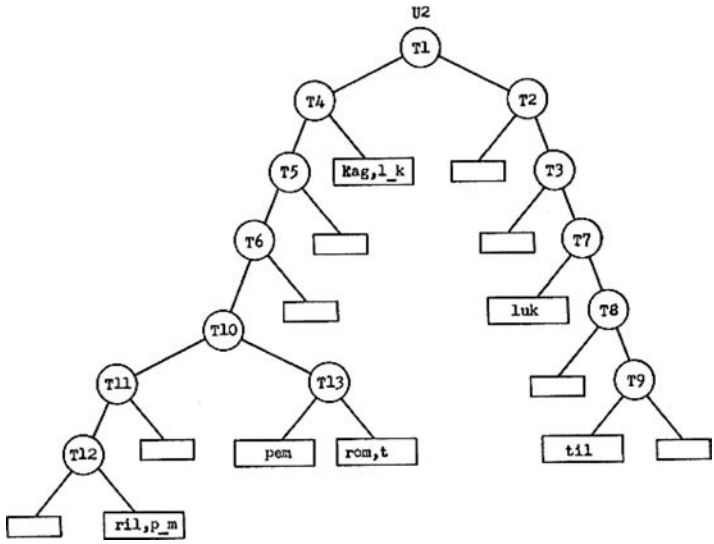

Figure 29.5. An EPAM discrimination net. (From Edward A. Feigenbaum, "An Information Processing Theory of Verbal Learning," Ph.D. dissertation, Carnegie Institute of Technology, p. 99, 1959, published as Report P-1817 by The RAND Corporation, Santa Monica, CA, October 9, 1959. Used with permission of Edward Feigenbaum.)

For AI purposes at least, the CLS systems were soon eclipsed by other decision-tree learning systems, namely, ID3, CART, and related programs. I'll describe how ID3 works as a way of explaining the main ideas behind these programs.

C. ID3

J. Ross Quinlan (1943– ; Fig. 29.6) developed ID3,[16] an acronym for Iterative Dichotomizer, in the late 1970s while he was on sabbatical leave (from the University of Sydney) at Stanford. (The name derived from the fact that the program constructed decision trees by iteratively dividing sets of data records until they could be classified into one of two distinct categories. Later versions allowed classification into more than two categories, but the "D" persisted in the name.) Quinlan had previously been a Ph.D. student (the first, actually) in the Computer Science Department at the University of Washington, working under Earl Hunt. Quinlan explained the genesis of ID3 in an e-mail note to me:[17]

> I sat in on a course given by Donald Michie [also visiting Stanford at that time] and became intrigued with a task he proposed, namely, learning a rule for deciding the result of a simple chess endgame. ID3 started out as a recoding of Buz's [that is, Earl B. Hunt's] CLS, but I changed some of the innards (such as the criterion for splitting a set of cases) and incorporated the iterative approach that allowed ID3 to handle the then-enormous set of 29,000 training cases.

Here, in brief, is how ID3 would proceed to construct a decision tree for predicting the value of the response attribute using my fictitious Wal-Mart database. First, ID3 would look for that single attribute to use as the "best" test in distinguishing between

Figure 29.6. J. Ross Quinlan. (Photograph courtesy of Ross Quinlan.)

those data records having the value yes for the response attribute from those having the value no. (I will have more to say about how "best" is defined momentarily.) No single test separates the data perfectly, but let us suppose that location does better than the others. After all, in this example *all* of the data records having value rural for the attribute location have the value yes for the response attribute, and none of those have the value no. Let's assume that the preponderance (but not all) of the data records having the value suburban have the value yes for the response attribute and that the preponderance (but again not all) of the data records having the value urban have the value no for the response attribute. Thus, the location attribute does a pretty good (but imperfect) job of separating data records with respect to the response attribute. A test for the value of the location attribute would thus be used as the first test in the decision tree being constructed.

So far then, we would have split the database into three subsets, two of which have data records with mixed values for the response attribute. ID3 would then apply the same splitting technique to each of these two mixed-value subsets, finding for each one of them the best next feature to use as a test. In this simple and rather nonrealistic example, the two tests that would be used, namely, type and customer, would each produce "pure" splits (that is, ones with no mixed values), and we would end up with the decision tree already shown in Fig. 29.4.

If the splits were not pure or not otherwise acceptable, however, ID3 would have gone on selecting tests on the resulting subsets of databases until the splits did give either pure or acceptable results.

The choice of which attribute to test on is critical in producing useful decision trees. In his original ID3 program,[18] Quinlan used a measure related to the "accuracy" of the resulting split in determining which attribute to use for testing. In later work, he used a measure called "information gain,"[19] whose precise definition I won't go into here except to say that it is that attribute whose values convey the most "information" about the categorization being sought. Quinlan used Claude Shannon's definition for measuring the amount of information.[20] Still later, he used a normalized measure of information gain in order not to bias in favor of tests with many outcomes.[21]

In my discussion of expert systems in Section 18.2, I mentioned that they were based on IF–THEN rules. Interest in symbolically based machine learning by Quinlan and others was mainly directed at learning these sorts of rules from data. It is quite easy to construct rules from a decision tree by tracing down the tests to generate the "IF" part and using what lies at the tips for the "THEN" part. For example, in the Wal-Mart database example, we could derive the following rules from the decision tree:

IF (location = suburban) and (type = ranch), THEN (response = no)
IF (location = suburban) and (type = multi-story), THEN (response = yes)
IF (location = rural), THEN (response = yes)
IF (location = urban) and (customer = yes), THEN (response = no)
IF (location = urban) and (customer = no), THEN (response = yes)

In Quinlan's work at Stanford, ID3 was able to generate rather large decision trees, and thus rule sets, for predicting whether certain endgame chess positions would end in a loss for black. For a problem of this type suggested by Donald Michie, ID3 used twenty-five attributes (involving features of the positions of pieces on the board) and a database of 29,236 different piece arrangements to construct a tree with 393 nodes whose predictions were 99.74% correct.[22]

One problem that must be avoided in constructing decision trees is that of "overfitting," that is, selecting tests based on so little data that the test results don't capture meaningful relationships in the data as a whole. No matter how large the original database, if a succession of tests eventually produces a subset that is still not pure but has been reduced to too few data records, any attempt to split that subset would overfit the data and thus not be useful. For that reason, decision-tree learning techniques typically halt tree construction just before data subsets would have too few records but would still give acceptable results.

D. C4.5, CART, and Successors

Quinlan continued his work on decision-tree-constructing systems, improving their power and applicability. He told me that "ID3 was pretty simple – about 600 lines of PASCAL."[23] His system C4.5 (which had about 9,000 lines of C) could work with databases whose attributes had continuous numerical values in addition to categorical ones. It could even deal with databases some of whose records had missing values for some of their attributes. Finally, it had methods for improving overall performance by pruning away some parts of the tree and for simplifying IF–THEN rules derived from trees.[24] A commercial company Quinlan founded in 1983 markets an improved version of C4.5 called C5.0 (along with a Windows version called See5).[25] Donald Michie also founded a company,[26] which independently developed a commercial version of ID3 called ACLS.

One of the significant developments in machine learning during this period was a fruitful collaboration between AI people and statisticians who were doing foundational as well as applied research on classification, estimation, and prediction. Each group has learned from the other, and machine learning is much richer for it. Although several people were involved in this collaboration, I might mention in

particular the Stanford statistician Jerome Friedman (1939–), who began working with some Stanford AI Ph.D. students in the 1990s. Following his earlier work on decision-tree construction,[27] Friedman, in collaboration with Leo Breiman, Richard Olshen, and Charles Stone, had helped develop a system called CART, an acronym for Classification and Regression Trees.[28] CART shares many features with C4.5 (and, in fact, C4.5 used CART's techniques for dealing with numeric attributes). At the time of this writing, the latest version, CART 5, is available from a commercial company.[29]

Systems for learning decision trees have been applied to a wide variety of data-mining problems.

E. Inductive Logic Programming

Expressed in the language of propositional logic, the IF–THEN rules produced from decision trees have the form $P1 \wedge P2 \wedge \ldots PN \rightarrow Q$. The P's and Q's are propositions with no internal structure. Earlier, I spoke of the predicate calculus in which propositions, called predicates, had internal arguments. In that language, one could have much more expressive rules such as $\forall(x, y, z)[\mathtt{Father}(x, y) \wedge \mathtt{Sibling}(z, y) \rightarrow \mathtt{Father}(x, z)]$, for example. Several techniques have been developed to learn these types of "relational" rules from databases and from other "background knowledge." (I mentioned a related topic before, namely, learning "probabilistic relational models," which are versions of Bayesian networks that permitted predicates with variables.) One of the early systems for learning relational rules was developed by Quinlan and called FOIL.[30] Because the rules learned have the same form as do statements in the computer language PROLOG (a language based on logic), the field devoted to learning these rules is called "Inductive Logic Programming" (ILP). Although ILP methods involve logical apparatus too complex for me to try to explain here, some of them bear a close relationship to decision-tree construction.[31] There are several applications of ILP, including learning relational rules for drug activity, for protein secondary structure, and for finite-element mesh design. These are all examples of what can be called "relational data mining."[32]

29.4 Neural Networks

During the 1960s, neural net researchers employed various methods for changing a network's adjustable weights so that the entire network made appropriate output responses to a set of "training" inputs. For example, Frank Rosenblatt at Cornell adjusted weight values in the final layer of what he called the three-layer alpha-perceptron. Bill Ridgway (one of Bernard Widrow's Stanford students) adjusted weights in the first layer of what he called a MADALINE. We had a similar scheme for adjusting weights in the first layer of the MINOS II and MINOS III neural network machines at SRI. Others used various statistical techniques to set weight values. But what stymied us all was how to change weights in more than one layer of multilayer networks. (I recall Charles Rosen, the leader of our group, sitting in his office with yellow quadrille tablets hand-simulating his ever-inventive schemes for making weight changes; none seemed to work out.)

29.4.1 *The Backprop Algorithm*

That problem was solved in the mid-1980s by the invention of a technique called "back propagation" (backprop for short) introduced by David Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams.[33] The basic idea behind backprop is simple, but the mathematics (which I'll skip) is rather complicated. In response to an error in the network's output, backprop makes small adjustments in all of the weights so as to reduce that error. It can be regarded as a hill-climbing (or rather hill-descending) method – searching for low values of error over the landscape of weights. But rather than actually trying out all possible small weight changes and deciding on that set of them that corresponds to the steepest descent downhill, backprop uses calculus to precompute the best set of weight changes.

Readers who remember a bit of college (or perhaps high school) calculus will have no trouble recalling that it can be used to calculate the slope of a curve or surface. The error in the output of a neural network can be thought of as a function of the network's weights, that is, a surface in "weight space." This function can be written down and "differentiated" (an operation in calculus) with respect to the weights to yield the set of weight changes that will take us downhill in the steepest direction. The problem with implementing this idea in a straightforward fashion for neural networks lies in the fact that these networks have "thresholds," whose effect is to populate the error surface with abrupt "cliffs." (The outputs of a network with thresholds can change from a 1 to a 0 or from a 0 to a 1 with infinitesimally small changes in some of the weight values.) Calculus operations require smoothly changing surfaces and are frustrated by cliffs.

Rumelhart and colleagues dealt with this problem by replacing the thresholds with components whose outputs can only change smoothly, even though they change quite steeply enough for the network to do approximately the same thing as a network with thresholds. With these replacements, calculus can be used to propagate the error function backward (from output to input) through the network to calculate the best set of changes to the weight values in all of the network's layers. Although this process of zeroing in on acceptable weight values is slow, it has been used with impressive results for many neural-network learning problems.

Why didn't *we* think of that? Actually, some people apparently did think of a similar idea before Rumelhart and his colleagues did. The earliest was probably Arthur E. Bryson Jr. and Y. C. Ho who used iterative gradient methods for solving Euler–Lagrange equations.[34] Paul Werbos, in his Harvard Ph.D. thesis, also proposed back-propagating errors to train multilayer neural networks.[35]

As with all local search techniques, backprop might get stuck on one of the local minima of the error surface. Of course, the learning process can be repeated, starting with different initial values of the weights, to attempt to find a lower (or perhaps the lowest) error value. In any case, the backprop method still is, as Laveen Kanal wrote in 1993, "probably the most widely used general procedure for training neural networks for pattern classification."[36]

Neural network learning methods have been applied in a variety of areas including aircraft control, credit card fraud detection, vending machine currency recognition, and data mining. I'll describe a couple of other applications next.

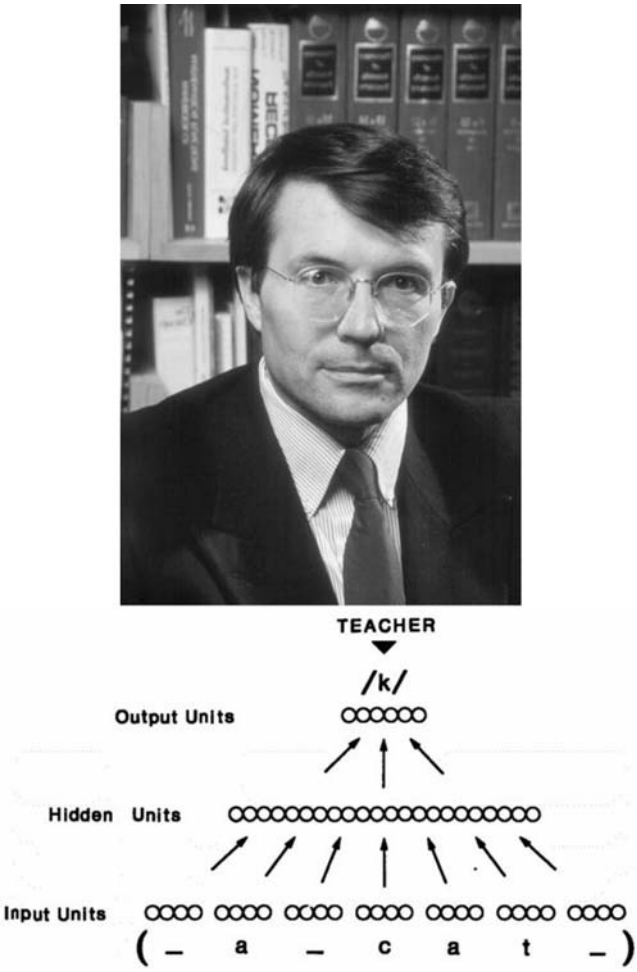

Figure 29.7. Terrance Sejnowski (top) and the neural network used in NETtalk (bottom). (Photograph and illustration courtesy of Terrance Sejnowski.)

29.4.2 *NETtalk*

One very interesting application of the backprop learning method was developed by Terrence J. Sejnowski (1947–) and Charles Rosenberg (1961–). They taught a neural network to talk![37] In one of their experiments, their system, called NETtalk, learned to "read" text that was transcribed from informal, continuous speech of a six-year-old child and produced acoustic output (that sounded remarkably like that of a child). (You can listen to an audio demo at http://www.cnl.salk.edu/ParallelNetsPronounce/.) The network structure is shown in Fig. 29.7.

The network had 203 input units designed to encode a string of seven letters. Text was streamed through these seven units letter by letter. There were 80 "hidden units" that were connected to the inputs by adjustable weights. It was hoped that

the hidden units would "form internal representations that were appropriate for solving the mapping problem of letters to phonemes." There were 26 output units that were supposed to produce coded versions of phonemes, the basic units of speech sounds. The output units were connected to the hidden units by additional adjustable weights. (Altogether, there were 18,629 adjustable weights.) Finally, the phonemic codes were fed to a commercial speech synthesizer to produce audible output.

The network was trained by comparing, at every time step, the phonemic code at the output units against what that code should have been for the text input at that time step. Backprop was used to modify the weights in a way that tended to reduce this error. The authors claim that "it proved possible to train a network with a seven letter window in a few days." (Remember that computers were much slower in 1987.) They concluded that "overall, the intelligibility of the speech was quite good" and that "the more words the network learns, the better it is at generalizing and correctly pronouncing new words." After training on a corpus of 1,024 words, the network "was tested [without further training] on a 439 word continuation from the same speaker. The performance was 78%, which indicates that much of the learning was transferred to novel words even after a small sample of English words." In addition to the specific network shown in Fig. 29.7, experiments were also done on networks with more hidden units and with two layers of hidden units. In general, the larger networks performed better.

29.4.3 *ALVINN*

Another neural network application, this one for steering a van, was developed by Dean Pomerleau, a Ph.D. student at Carnegie Mellon University.[38] The system, which included the van, a TV camera for looking at the road ahead, and interface apparatus, was called ALVINN, an acronym for Autonomous Land Vehicle in a Neural Network. ALVINN used the CMU Navlab vehicle, which was built on a commercial van chassis with hydraulic drive and electric steering. According to a CMU paper, "Computers can steer and drive the van by electric and hydraulic servos, or a human driver can take control to drive to a test site or to override the computer."[39] A picture of Navlab is shown in Fig. 29.8.

The input to ALVINN's neural network was a low-resolution 30×32 array of gray-scale image intensity values produced by a video camera mounted on top of the van. Each of these 960 inputs was connected to each of four hidden units through adjustable weights. The hidden units, in turn, were connected to a left-to-right line of 30 output units through adjustable weights. The output units controlled the van's steering mechanism as follows:[40]

> The centermost output unit represents the "travel straight ahead" condition, while units to the left and right of center represent successively sharper left and right turns. The units on the extreme left and right of the output vector represent turns with a 20 m radius to the left and right respectively, and the units in between represent turns which decrease linearly in their curvature down to the "straight ahead" middle unit . . .
>
> The steering direction dictated by the network is taken to be the center of mass of the "hill" of activation surrounding the output unit with the highest activation level. Using the center

Figure 29.8. CMU's Navlab vehicle used by ALVINN. (Photograph courtesy of Carnegie Mellon University.)

of mass of activation instead of the most active output unit when determining the direction to steer permits finer steering corrections, thus improving ALVINN's driving accuracy.

Figure 29.9 shows the arrangement of the network and a typical low-resolution road image as presented to the network.

There were various versions of ALVINN. In one, training of the network was "on-the-fly," meaning that the network was trained in real time as the van was steered by a human driver along various roads and paths. The desired steering angle was the

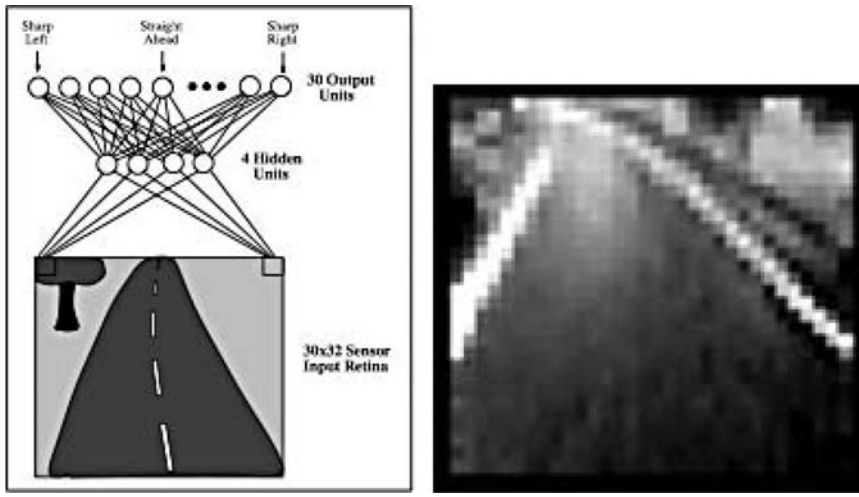

Figure 29.9. The ALVINN network (left) and a typical road image (right). (From Dean A. Pomerleau, "Neural Network Vision for Robot Driving," Michael Arbib (ed.), *The Handbook of Brain Theory and Neural Networks*, Cambridge, MA: MIT Press, 1995. A version of this paper is available online at http://www.ri.cmu.edu/pub_files/pub2/pomerleau_dean_1995_1/pomerleau_dean_1995_1.pdf.)

one selected by the driver, and the network weights were adjusted by backprop to attempt to mimic the driver's performance. One problem with this method was that the network was never exposed to possible "going-off-the-road" images. Simulations of what such images would look like (labeled by what the steering angle should be in those cases) were added to the training set.

In summarizing a typical test of ALVINN's performance, Pomerleau wrote[41]

> Over three runs, with the network driving at 5 miles per hour along the 100 meter test section of road, the average position of the vehicle was 1.6 cm right of center, with a standard deviation of 7.2 cm. Under human control, the average position of the vehicle was 4.0 cm right of center, with a standard deviation of 5.47 cm.

Carnegie Mellon's Robotics Institute continued (and still continues) to work on autonomous vehicles, although the neural-network approach to image-guided steering was replaced by more robust computer-vision algorithms. Their 1995 visual perception system RALPH (an acronym for Rapidly Adapting Lateral Position Handler) used special image-processing routines to determine road boundary curvature. According to Pomerleau,[42] "RALPH has been able to locate the road and steer autonomously on a wide variety of road types under many different conditions. RALPH has driven our Navlab 5 testbed vehicle over 3000 miles on roads ranging from single lane bike paths, to rural highways, to interstate freeways."

In the summer of 1995, one of their specially outfitted vehicles, a 1990 Pontiac Trans Sport (Navlab 5) donated by Delco Electronics, steered autonomously (using RALPH) for 2,797 of the 2,849 miles from Pittsburgh, PA to San Diego, CA. (Only the steering was autonomous – Pomerleau and Ph.D. student Todd Jochem handled the throttle and brake.) The average speed was above 60 miles per hour.[43]

29.5 Unsupervised Learning

The decision tree and neural network learning methods described so far in this chapter are examples of "supervised learning," a type of learning in which one attempts to learn to classify data from a large sample of training data whose classifications are known. The "supervision" that directs learning in these systems involves informing the system about the classification of *each* datum in the training set. Yet, it is sometimes possible to construct useful classifications of data based just on the data alone. Techniques for doing so fall under the heading of "unsupervised learning."

Recall that in Section 4.3 I showed a diagram (Fig. 4.11) in which data to be classified were represented by points in a two-dimensional "feature space." The coordinates of the points corresponded to the values of two numerically valued features, f_1 and f_2, of the data. In Fig. 4.11, the category of each point was indicated by small squares for points belonging to one category and small circles for points belong to another category. Because the points were thus labeled, they could be used as training examples for a supervised learning procedure.

But suppose we have a set of unlabeled sample points, such as those shown in Fig. 29.10. Can anything be learned from data of that sort? By visual inspection, we see that the points seem to be arranged in three clusters. Perhaps each cluster contains points that could be thought to belong to the same category. So, if we could

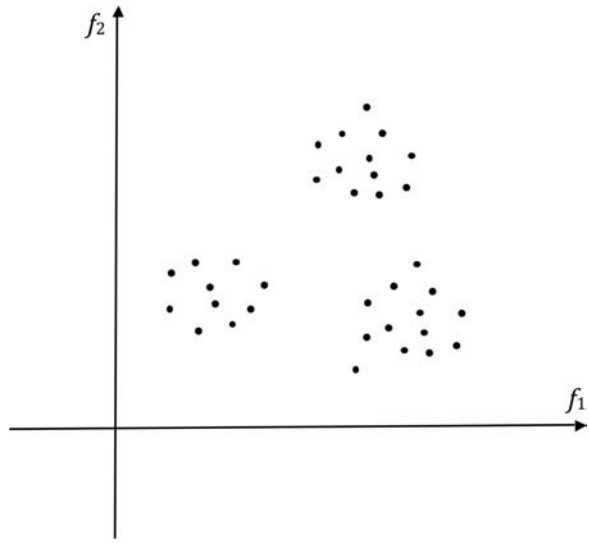

Figure 29.10. Unlabeled points in a feature space.

automatically process data samples to identify clusters and the boundaries between them, we would have a method of unsupervised learning.

AI researchers have used several methods for identifying clusters of training samples. A popular one, and one that is easy to explain, is the so-called *k*-means method. It works by repeating over and over the following steps:

1. Install, perhaps at random locations, some number, say *k*, of "cluster seekers" in the space of samples.
2. For each of these cluster seekers, group together those training samples that are closer to it than to any other cluster seekers.
3. Compute the centroid (the "center of gravity") of each of these groups of samples.
4. Move each of the cluster seekers to the centroid of its corresponding group.
5. Repeat these steps until none of the cluster seekers needs to be moved again.

At the end of this process, the cluster seekers will all be at the centroids of groups of training samples that can be considered to be clusters or separate categories of data. Now to classify some new data point not in the training set, we simply compute to which cluster seeker it is closest. The process depends, of course, on being able to guess the number of clusters, *k*. Methods for doing so generally involve adjusting the number of them so that points within clusters are closer together than the distances between clusters.

Statisticians and others have developed several methods for clustering data, including variations related to the *k*-means method. One prominent technique, AutoClass, was developed by Peter Cheeseman and colleagues at NASA.[44] According to a Web site about AutoClass,[45]

AutoClass takes a database of cases described by a combination of real and discrete valued attributes, and automatically finds the natural classes in that data. It does not need to be told how many classes are present or what they look like – it extracts this information from the data itself. The classes are described probabilistically, so that an object can have partial membership in the different classes, and the class definitions can overlap.

AutoClass is famous for having discovered a new class of infrared stars. It has also discovered new classes of proteins, introns, and other patterns in DNA/protein sequence data.

There are even techniques that can be applied to non-numeric data. Statisticians group all of these methods (numeric and non-numeric) under the general heading of "cluster analysis." A good overview can be found in the online Electronic Textbook *StatSoft* at http://www.statsoft.com/textbook/stcluan.html. The textbook by Duda, Hart, and Stork has a thorough discussion of unsupervised learning (as well as other topics in data classification).[46]

29.6 Reinforcement Learning

29.6.1 *Learning Optimal Policies*

There is another style of learning that lies somewhat in between the supervised and unsupervised varieties. An example would be learning which of several possible actions a robot, say, should execute at every stage in an ongoing sequence of experiences given only what final result of all of its actions. An extreme case would be learning to play excellent chess given only information about a win or a loss at the end of play. No system has yet been built that can learn to play chess that way, but it is possible for a program to learn to play backgammon that way and to learn to perform other interesting tasks, such as controlling the flight of helicopters. Borrowing terms from psychological learning theory, we can call the win or loss information (or in general the good-result or bad-result information) a "reward" or a "reinforcement," and this style of learning is called "reinforcement learning" or (sometimes) "trial-and-error learning."

Reinforcement learning has a long and varied history. The psychologist Edward L. Thorndike (1874–1949) studied this style of learning in animals.[47] In their book *Reinforcement Learning: An Introduction*,[48] Richard S. Sutton (1957– ; Fig. 29.11) and Andrew G. Barto (1948– ; Fig. 29.11), two of the field's pioneers, mention some additional historical milestones, including Arthur Samuel's method for learning evaluation functions in checkers, the use of Richard Bellman's dynamic programming techniques in optimal control, John Andreae's trial-and-error learning system STeLLA,[49] Donald Michie's learning systems for tic-tac-toe (MENACE[50]) and pole-balancing (BOXES[51]), and A. Harry Klopf's work on "hedonistic neurons."[52] Reinforcement learning is another one of those subdisciplines of AI that has become highly technical and multibranched. I'll attempt a gentle and nonmathematical description of how it works.

In its simplest setting, reinforcement learning is about learning how to traverse a collection of states, going from one state to another and so on, to reach a state in

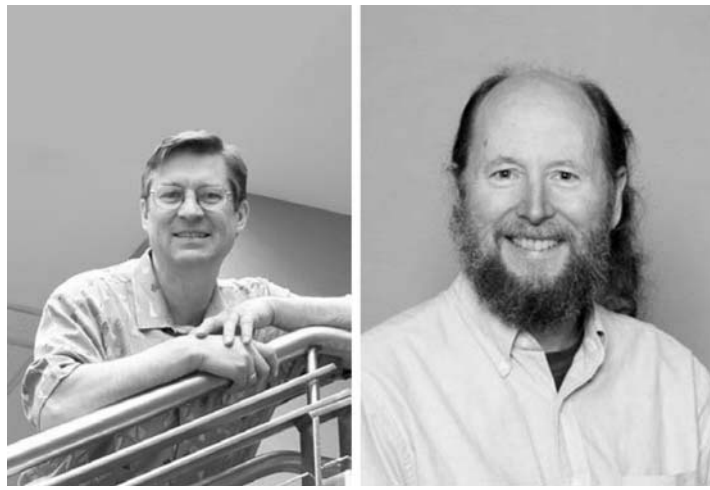

Figure 29.11. Andrew Barto (left) and Richard Sutton (right). (Photographs courtesy of Andrew Barto and of Richard Sutton.)

which a reward is obtained. The problem is much like one that a rat faces in learning how to run a maze (or one that a robot faces in learning how to carry out a task). In fact, let us use a maze example to describe some of the aspects of reinforcement learning. A typical maze is shown in Fig. 29.12.

The rat's problem is to go from its starting position to the cheese at the goal position. The gray dots in the figure are meant to depict situations that the rat might find itself in and recognize. In reinforcement learning terminology, these situations are called "states." At each state, the rat can select from among, say, four actions, namely, turn left, turn right, go forward, or go back. Depending on the state, only some of the actions are possible – one cannot go forward when up against a dead end for example. Each possible action takes the rat from one state to an adjacent one in the maze. The collection of states and the actions that link them can be thought of as a graph, similar to those I discussed when I talked about search methods.

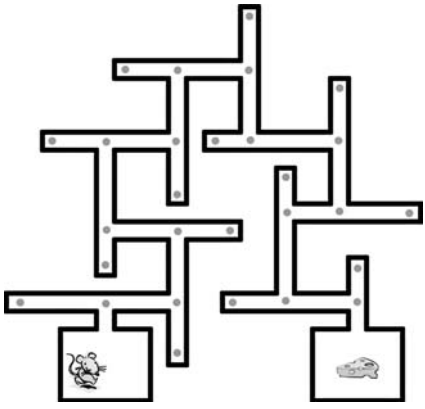

Figure 29.12. A maze.

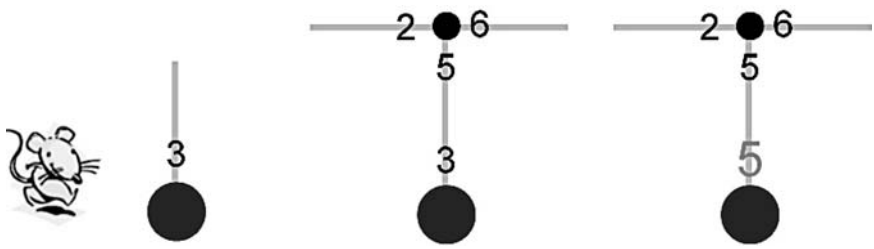

Figure 29.13. Initial stages of the Q-learning process.

So as not to stray too far from what is known about real rats running mazes, let us switch now to describe how a fictional "robotic rat" might learn how to run this maze. The main problem for the robot is that it starts out by not having a map of the maze nor having any idea about the effects of its actions. That is, for any given state that it finds itself in, it does not know which next states would result for the various actions it could take in that state. For if it did have such a map, say one represented by a graph, it could search the graph (using a method like A*) to find a path to the goal node. One way to proceed would be to attempt to learn a graph of states and their connections by trial-and-error methods and then to use graph-searching methods to figure out how to navigate the maze.

An alternative, and the one used by most reinforcement learning methods, involves naming all of the states that the robot encounters as it wanders randomly in search of the goal. (We assume that eventually it does reach the goal.) In reinforcement learning terminology, a "policy" for running the maze associates some single action with each named state. A best or "optimal policy" would associate with each state that action that would lead to a shortest (or otherwise least costly) path through the maze. Reinforcement learning is about learning the best policy, or, at least, good policies.

One method for learning a policy involves associating a "valuation" number with every possible action at each state and then adjusting these numbers (based on experience) until they point the way toward the goal. This method is called "Q-learning" and was originally suggested by Christopher Watkins (1959–) in his Cambridge University Ph.D. thesis.[53] The robot begins its learning process by assigning a name to the state in which it begins and by assigning randomly selected valuation numbers to every action it can take in that state. The learning process will expand this table by assigning names and valuation numbers to all of the actions it can take in every *new* state encountered. (We assume that the robot remembers, in its table, the names of all the states it has already visited in its learning process and can distinguish these from new states.) The robot's initial state, with a randomly selected valuation number assigned to its only action possible, is shown in the left-hand sketch of Fig. 29.13. At every stage of the learning process, the robot takes that action having the highest valuation number. Because there is only one action in the robot's initial state, it takes that action, finds itself in a new state, and assigns random valuation numbers to the actions possible in that new state. This step is shown in the middle sketch of Fig. 29.13. Now comes the key step in learning. Because the robot

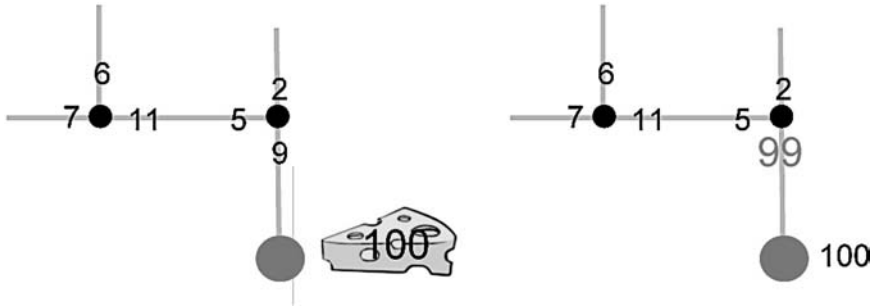

Figure 29.14. Stages leading to the goal.

now "knows" that it can reach a new state having actions whose highest valuation number is 6, it updates the valuation number, namely, 3, of the action leading to that state by adjusting it to a number more consistent with being able now to take an action that it imagines is worth 6. To account for the "cost" of its just-completed action, the adjustment of 3 does not go all the way to 6 but just to 5, say. The result is shown in the right-hand sketch of Fig. 29.13 in which the adjusted valuation is shown a bit larger than the other numbers and shaded.

This process continues. In each state, take that action whose valuation number is largest and then adjust that valuation number by making its value closer to the value of the action with the highest valuation number in the state just entered. And, even though the process starts with randomly selected valuation numbers, eventually the trial-and-error process will stumble into the goal state where a high "reward" will be obtained. At that stage, the action just taken, which led to that reward, has its valuation number raised to the same value (or maybe just a little bit less) than the value of the reward. I illustrate this step in Fig. 29.14. The sketch on the left of the figure shows some of the states and action valuations at the time the robot takes the action that achieves the goal. In the sketch on the right of the figure, I show the adjusted valuation (shaded) for that goal-achieving action. Now, for the first time, an action valuation is based on getting a reward rather than being set randomly. If the robot ever finds itself in the state adjacent to the goal state again, it will certainly take the same action. More importantly, when it reaches this penultimate state in a subsequent experience, it will propagate this reward-based value backward.

I illustrate how backward propagation works in Fig. 29.15. Suppose, in the sketch on the left, the robot finds itself in the state marked by an arrow. From that state, it takes that action with the largest valuation, which leads it to a state adjacent to the goal. The action with the largest valuation leading out of that state has a valuation of 99, so the valuation of the action just taken is changed from 11 to 98, as shown in the sketch on the right. Increasing the valuations of actions in states close to the goal by backward propagation, in effect, makes those states intrinsically "rewarding" just as if they were goal states themselves.

The astute reader may complain that I have cleverly set the "random" valuation numbers to values that would lead to the goal once the robot gets to states close to

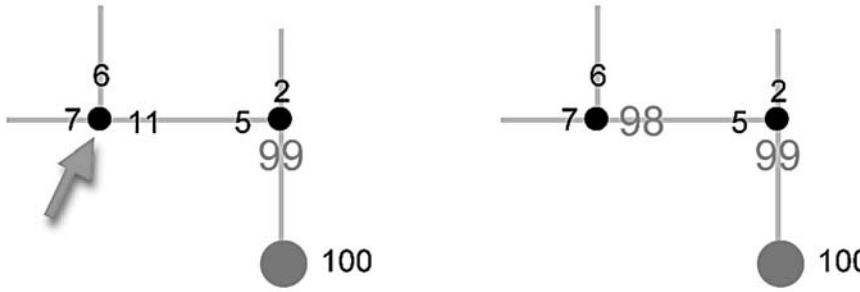

Figure 29.15. Backward propagation of goal-achieving action valuations.

the goal. What if these values were such, as they most probably would be, that, upon getting close, the robot wanders away from the nearly achieved goal? If the valuation numbers are adjusted as I prescribed, always taking into account the cost of a move, a little thought will convince one that eventually the numbers will be such as to force the robot toward the goal, with all other avenues eventually being closed off.

With continued experience, the valuations of actions involved in achieving the goal gradually propagate backward from the goal. Eventually, after much trial-and-error experience (and with some "reasonable" assumptions), the values will converge to those that implement an optimal policy, that is, one that always gets the robot to the goal in the most efficient manner.

Most versions of reinforcement learning have the following elaborations:

- Rewards might be given at more than one of the states. That is, there is not necessarily a single goal state but many states that might contribute to reward. Rewards are represented by numerical values, which could be positive (true "rewards"), zero, or negative ("punishments").
- Rather than attempting to find a policy that corresponds to an optimal path to a single goal state, one tries to learn policies that maximize the amount of reward expected over time. Usually in learning a policy, rewards that are anticipated in the distant future are "discounted," that is, they don't count as much as rewards expected more immediately.
- Any given action taken at a state might not always lead to the same state. One could attempt to learn the probabilities that certain actions taken in a state lead to which other states, and some reinforcement learning methods, such as "prioritized sweeping,"[54] do that. The Q-learning process avoids the need to learn these probabilities explicitly because, whatever they are, they (along with rewards) appropriately affect the values that the learning process assigns to state–action pairs.
- As a further complication, it might be that the robot has only imperfect knowledge of what state it is in because its sensory apparatus is not sufficiently accurate or informative. In that case, the actual state that the robot is in is said to be "hidden" from it, which adds additional complications to the problem of learning an optimal policy.

With these elaborations, the problem becomes one of what is called a "Markov Decision Process" (MDP). With imperfect state knowledge, it is called a "Partially-Observable Markov Decision Process" (POMDP). MDPs and POMDPs have been well studied by people in control theory as well as in AI.[55]

I can use the robot maze example to mention several things that are important in the use of reinforcement learning in practical applications. First, I assumed that the robot's random exploration eventually would land it in the goal state. In complex problems, the chance of randomly achieving a goal (or other rewards) might be slim to none. Breaking the problem down into a hierarchy of subproblems in which rewards are more easily obtained is sometimes used to speed up learning. Additionally, "shaping" strategies can be used in which the robot is first placed in a situation sufficiently close to the goal that random exploration will find the goal. Then, after some actions close to the goal have been assigned goal-relevant evaluations, the starting situations can be gradually moved farther and farther from the goal. Alternatively, hints might be given, perhaps in the form of intermediate rewards given to let the robot know that it is doing well so far. Strategies such as these are used in teaching skills to humans and animals.

Another problem concerns the tradeoff between "exploiting" an already learned policy versus "exploring" to find better policies. It is often the case that a set of action valuations obtained early in the learning process might not be the best set possible. To learn a better set, the robot must be encouraged in some way to strike out randomly away from a known policy to lock on to a better one. Finally, many problems might have "state spaces" so large that the entire set of all of the states and their actions and valuations cannot be explicitly listed in a table like the one I assumed for the robot maze problem. In that case, the valuations of actions that can be taken in a state must be computed rather than stored. I'll show an example of how that might be done in the next few pages.

29.6.2 *TD-GAMMON*

One of the most impressive demonstrations of the power of machine-learning methods is the TD-GAMMON system developed by Gerald Tesauro at IBM.[56] Versions of TD-GAMMON learned to play excellent backgammon after playing against themselves during millions of games. TD-GAMMON used a combination of neural net learning and a type of reinforcement learning called "temporal difference learning" (which explains the prefix TD).

TD-GAMMON's neural network consisted of three layers. In one version there were 198 input units, 40 hidden units, and 4 output units. Each of the output units could have an output value between 0 and 1. (Instead of threshold units, the network had the kinds of components I talked about earlier, namely, those whose outputs changed smoothly, but still abruptly, between 0 and 1.) Each of the outputs was charged with the task of estimating a probability of a particular outcome of the game. The four possible outcomes considered were white wins, white gammons, black wins, or black gammons. The input units were coded to represent the configuration of pieces on the board. The values of the four outputs were combined to yield a number giving the estimated "value" of a board position from white's point of view.

I'll describe how the network learned in a moment. First, here is how the network was used to select a move. (I'm assuming here that the reader has some familiarity with backgammon, but my description should make sense even for those who do not.) At each stage of play, the dice are thrown, and the program considers all of the possible moves that it might make given that throw of the dice. The network computes the value of each possible resulting board, and the program selects the move producing the board with the best value (which is the highest value when it is white's move and the lowest value when it is black's move).

Now, here's how the network learns: For each board position encountered during actual play, the network's weights are adjusted, using backprop, so that the value computed for that board position is closer to the value computed for the temporally next board position (and thus we see why the term "temporal difference" arises). The network starts with randomly selected weight values, so the moves early in the learning process, as well as the weight adjustments, are random. But eventually, even randomly selected moves result in a win for one of the players. After a win occurs, the four probability values are then known for sure – one of them is "1," and the rest are "0." The network's weights can then be adjusted so that the value of the penultimate board is made closer to the value of this final, winning board position. As in all reinforcement learning procedures, values are gradually propagated backward from the end of the game toward the starting position. After millions of games, the network weights take on values that result in expert play. In commenting on a version of TD-GAMMON that uses search in addition to learning, Sutton and Barto wrote[57]

> TD-GAMMON 3.0 appears to be at, or very near, the playing strength of the best human players in the world. It may already be the world champion. These programs have also already changed the way the best human players play the game. For example, TD-GAMMON learned to play certain opening positions differently than was the convention among the best human players. Based on TD-GAMMON's success and further analysis, the best human players now play these positions as TD-GAMMON does.

29.6.3 *Other Applications*

There are probably hundreds of important applications of reinforcement learning methods. A typical, as well as dramatic, example is the work of Andrew Ng (1976–) and his group at Stanford on learning to perform aerobatic helicopter maneuvers.[58] Some photographs of a model helicopter that has learned to "roll" are shown in Fig. 29.16. Other applications have been in elevator dispatching, job-shop scheduling, managing power consumption, and four-legged walking robots.

As a final comment about reinforcement learning, it is interesting to observe that part of the technology of machine learning, a part whose name was borrowed from psychology, now pays back its debt by providing a theoretical framework for how animal brains learn at the neurophysiological level. In an article in *The Journal of Neuroscience*, Christopher H. Donahue and Hyojung Seo wrote[59]

> To make effective decisions while navigating uncertain environments, animals must develop the ability to accurately predict the consequences of their actions. Reinforcement learning has emerged as a key theoretical paradigm for understanding how animals accomplish this feat . . .

Figure 29.16. Andrew Ng (top) and his model helicopter during a roll maneuver (bottom). (Photographs courtesy of Andrew Ng.)

In addition to successfully predicting the animal's choice behavior, the reinforcement learning model has been successfully used to elucidate the function of the basal ganglia in goal-directed behavior. Dopaminergic neurons in the ventral tegmental area and the substantia nigra have been shown to encode a reward-prediction error, which is used to improve the outcomes of an animal's future choices. Another study in monkeys engaged in a free-choice task showed that the activity of striatal neurons is correlated with action values, which were estimated by integrating the previous outcome history associated with each action.

29.7 Enhancements

Many of the machine learning methods I have mentioned can be enhanced in various ways. Some of these are based on work by statisticians and others by people working on what is called "computational learning theory." One technique, called "bagging" (an acronym for bootstrap aggregating) is due to Professor Leo Breiman of the University of California, Berkeley.[60] For classification problems, bagging works by combining the outputs of a number, say m, of separate classifiers. Each classifier is trained by using a different subset of the original training set. These subsets are obtained from the original by randomly selecting (with replacement) some of its examples. (Statisticians call these samples "bootstrap samples.") After each of the m classifiers is trained, final classification is made by a majority vote. The technique can be applied independently of the kind of individual classifier used – neural network, decision tree, nearest-neighbor, or what have you. Bagging can also be used for the problem of associating a number (rather than a category) with an example. In

that case, outputs are averaged rather than participating in a vote. The voting and averaging operations help avoid overfitting the data and thus yield better performance than would have been obtained with one classifier trained on all of the data. [One wonders how the performance of the 1960s MADALINE neural network (see p. 69) might have been improved had each of its threshold units been trained on bootstrap samples.]

A related idea, called "boosting," was proposed by Robert E. Schapire.[61] Although there are many versions, here in outline is how it works. Using any of the supervised machine learning methods, a classifier is trained on the original training set in which each sample is equally "weighted." (The ith sample's weight, say w_i, can be set, for example, by including that sample w_i times in the training set.) Then a new training set is constructed in which those samples that were misclassified have their "weights" increased, and those samples that were correctly classified have their weights decreased. Using this new training set, another classifier is trained. (That one will, presumably, work harder on the earlier misclassified samples.) This process is repeated until we have some number, say m, of classifiers. Now, each of the classifiers votes on the categorization of new samples. Their votes are weighted by how well they performed on the original training set. Votes of the more reliable classifiers count more than do those of less reliable classifiers. Even when the original classifiers are "weak" (that is, not very reliable at all), the overall accuracy of the combined set of m classifiers can be quite good, thus "boosting" the results.

Several ways of doing boosting have been proposed. One of the popular ones, due to Yoav Freund and Robert Schapire, is called "Adaboost."[62] It is also possible to combine bagging and boosting.[63]

Finally, I'll mention "Support Vector Machines" (SVMs). A complete description of them would involve more mathematics than we want to get into here, but I can give a rough-and-ready idea of how they work by using a geometric example. On the left-hand side of Fig. 29.17 I show the same points that I used in Fig. 4.11 to illustrate a separating boundary in feature space. The points indicated by small squares correspond to samples in one category, and the points indicated by small circles correspond to samples in another category.

As a reminder, the points in the diagrams have coordinates equal to the features, f_1 and f_2, computed from items (such as speech sounds, images, or other data) that we want to classify. It happens in this case that there exists many straight-line (that is, linear) boundaries that would separate the points in the two categories perfectly. Therefore an attempt to train a neural element to classify the points (considered as "training samples") would be successful. If we used the standard error-correction procedure for training, we would certainly get *some* linear boundary, but with SVMs we ask more of the boundary than that it merely separate the training samples. We want it to be such that the distances (called the "margin") from it to the closest points of opposite categories are as large as possible. Such a linear boundary is shown on the right-hand side of Fig. 29.17. The parallel dashed lines on either side go through these closest points, which are called "support vectors." Boundaries with margins as large as possible are desirable because they are better at classifying new points not in the training set. That is, they have better "generalizing" properties.

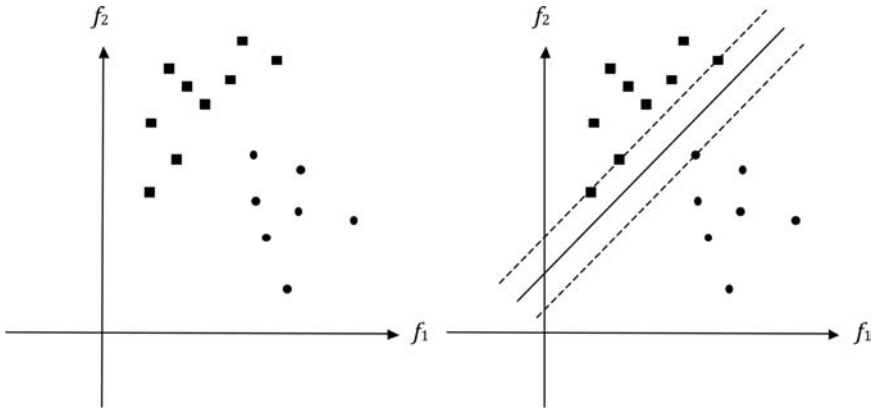

Figure 29.17. Points and a linear separating boundary in a two-dimensional space.

Our early work on pattern recognition (of the supervised learning variety) at SRI included some experiments in which we attempted to find separating boundaries that were insulated away from the training samples. One of the methods for doing so involved including training samples derived from the original ones by adding small amounts of "noise" to them. The idea was that the error-correction training procedure applied to this augmented set would be forced away from the original samples. A more elegant method was proposed by H. Glucksman, in which error-correction training continued until some minimum allowed distance between training samples and separating boundaries was achieved.[64] To ensure margins as large as possible, however, requires some complex optimization procedures. (Mathematically inclined readers can refer to an online tutorial by Tristan Fletcher at http://www.tristanfletcher.co.uk/SVMExplained.pdf or to a textbook by Nello Cristianini and John Shawe-Taylor about SVMs.[65])

Now, you might ask, how does one get feature spaces that are linearly separable? One way is to use something like Rosenblatt's alpha-perceptron. Recall that the elements in the alpha-perceptron's first layer of threshold elements, say N of them, each received its own input from a random collection of data measurements (such as pixels or speech waveform values). The binary outputs of these "association units" (as these first-layer elements were called) were then features like those I used in the two-dimensional example. They determined points in an N-dimensional feature space, which (Rosenblatt hoped) was linearly separable. Often, in Rosenblatt's work, they were.

The people working with SVMs use a different method for defining features. Their method ensures that the resulting feature space is linearly separable (or, at least, nearly so). Their features involve the use of what they call "kernels," and machines using such features are called "kernel machines." Again, the mathematics is too complex to be described here, but the interested reader can look at the book by Nello Cristianini and John Shawe-Taylor. As that book points out, the history of the mathematics leading up to kernel machines and SVMs goes as far back as the

beginning of the twentieth century and has involved people in optimization theory, statistics, and computational learning theory.

SVMs and kernel machines are superb examples of how work in several disciplines, using highly technical mathematical apparatus, has contributed to powerful new techniques in artificial intelligence. Important venues for describing new work in machine learning are the Neural Information Processing Systems (NIPS) Conferences sponsored annually by the Neural Information Processing Systems Foundation.[66]

After hearing about all of the methods for machine learning described in this chapter, you might reasonably ask, which method is best? Should one use the nearest-neighbor method, a decision tree, a neural network, or something else? Researchers have asked that question also, and there have been "bake-offs" in which different methods have competed on various standard problems, such as character recognition. One such competition, organized by the European Community ESPRIT project StatLog, is described in a book edited by Donald Michie, D. J. Spiegelhalter, and C. C. Taylor.[67] Another comparison of several methods was reported in the AI text by Russell and Norvig.[68] Some methods work better for some problems than for others, but often these differences are only marginal, and most people in the field agree that having lots and lots of data is, in the end, more important than the particular machine learning algorithm used. That is, spend time gathering more data rather than tuning a particular method.[69]

Notes

1. For a nice review, see the online tutorial put together by Andrew H. Moore, one of the pioneers of memory-based learning, at http://www.autonlab.org/tutorials/mbl08.pdf. [399]
2. Christopher G. Atkeson, Andrew W. Moore, and Stefan Schall, "Locally Weighted Learning for Control," *Artificial Intelligence Review*, Vol. 11, pp. 75–113, 1997; available online at http://www.cs.cmu.edu/~cga/papers/air1.ps.gz. See also Stefan Schaal and Christopher G. Atkeson, "Robot Juggling: An Implementation of Memory-based Learning," *IEEE Control Systems Magazine*, Vol. 14, No. 1, pp. 57–71, February 1994; available online at http://www-clmc.usc.edu/publications/S/schaal-CSM1994.pdf. [399]
3. Walter Daelemans and Antal van den Bosch, *Memory-Based Language Processing*, Cambridge: Cambridge University Press, 2005. [400]
4. Janet Kolodner, *Case-Based Reasoning*, pp. 18–19, San Francisco: Morgan Kaufmann Publishers, 1993. [400]
5. See Janet Kolodner, "Reconstructive Memory: A Computer Model," *Cognitive Science*, Vol. 7, No. 4, pp. 281–328, 1983, and Michael Lebowitz, "Memory-Based Parsing," *Artificial Intelligence*, Vol. 21, pp. 363–404, 1983. [401]
6. Edwina L. Rissland, unpublished notes. [401]
7. Edwina L. Rissland, "Example Generation," *Proceedings Third National Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 280–288, Victoria, BC, 1980. [401]
8. See, for example, Edwina L. Rissland, "Examples in the Legal Domain: Hypotheticals in Contract Law," *Proceedings Fourth Annual Cognitive Science Conference*, pp. 96–99, University of Michigan, Ann Arbor, 1982. [401]
9. E-mail of February 17, 2009. [401]

10. http://www.aiai.ed.ac.uk/technology/casebasedreasoning.html. [402]
11. Feigenbaum told me about another tree-construction system developed independently (and for different purposes) around 1959 by Edward Fredkin. See Edward Fredkin, "Trie Memory," *Communications of the ACM*, Vol. 3, No. 9, pp. 490–499, September 1960. [404]
12. Edward A. Feigenbaum, "The Simulation of Verbal Learning Behavior," *Proceedings of the Western Joint Computer Conference*, Vol. 19, pp. 121–132, 1961. Reprinted in Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, New York: McGraw-Hill, 1963. [404]
13. See, for example, Edward A. Feigenbaum and Herbert Simon, "EPAM-like Models of Recognition and Learning," *Cognitive Science*, Vol. 8, No. 4, pp. 305–336, 1984; Howard B. Richman, J. J. Staszewski, and Herbert A. Simon, "Simulation of Expert Memory Using EPAM-IV," *Psychological Review*, Vol. 102, No. 2, pp. 305–330, 1995; and Howard B. Richman, Herbert A. Simon, and Edward A. Feigenbaum, "Simulations of Paired Associate Learning Using EPAM-VI," Complex Information Processing Working Paper #553, Department of Psychology, Carnegie Mellon University, March 7, 2002. The latter paper is available online at http://www.pahomeschoolers.com/epam/cip553.pdf. [404]
14. Carl I. Hovland and Earl B. Hunt, "Programming a Model of Human Concept Formulation," *Proceedings of the Western Joint Computer Conference*, pp. 145–155, May 9–11, 1961. Reprinted in Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, pp. 310–325, New York: McGraw-Hill, 1963. [404]
15. Earl B. Hunt, Janet Marin, and Philip J. Stone, *Experiments in Induction*, New York: Academic Press, 1966. [404]
16. For Quinlan's descriptions of ID3, see J. Ross Quinlan, "Discovering Rules by Induction from Large Collections of Examples," in Donald Michie (ed.), *Expert Systems in the Micro Electronic Age*, pp. 168–201, Edinburgh: Edinburgh University Press, 1979, and J. Ross Quinlan, "Induction of Decision Trees," *Machine Learning*, Vol. 1, pp. 81–106, 1986. Available online at http://www.cs.toronto.edu/~roweis/csc2515-2006/readings/quinlan.pdf. [405]
17. From an e-mail from Quinlan to me dated March 18, 2008. [405]
18. That is, the one described in J. Ross Quinlan, *op. cit.* [406]
19. See, for example, J. Ross Quinlan, "Learning Efficient Classification Procedures and Their Application to Chess End Games," Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell (eds), *Machine Learning: An Artificial Intelligence Approach*, pp. 463–482, San Francisco: Morgan Kaufmann Publishers, 1983. (By the way, the very title of that volume indicates, I think, that the editors wanted to contrast the approach used in the volume's papers with neural network approaches to machine learning.) [406]
20. The classic paper is Claude E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, pp. 379–423 and 623–656, July and October 1948. Available online at http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf. [406]
21. See, for example, J. Ross Quinlan, "Decision Trees and Multi-Valued Attributes," in Jean E. Hayes, Donald Michie, and J. Richards (Eds.), *Machine Intelligence 11*, pp. 305–318, Oxford: Oxford University Press, 1988. All of these measures are described in J. Ross Quinlan, *C4.5: Programs for Machine Learning*, San Francisco: Morgan Kaufmann Publishers, 1993. [406]
22. J. Ross Quinlan, "Discovering Rules by Induction from Large Collections of Examples," in Donald Michie (ed.), *Expert Systems in the Micro Electronic Age*, pp. 168–201, Edinburgh: Edinburgh University Press, 1979. [407]
23. E-mail communication, March 18, 2008. [407]

24. See J. Ross Quinlan, *C4.5: Programs for Machine Learning*, San Francisco: Morgan Kaufmann Publishers, 1993. **[407]**
25. A complete version of C4.5 can be downloaded free of charge from Ross Quinlan's homepage, http://www.rulequest.com/Personal/. Scaled-down versions of C5.0 and See5 can be downloaded free of charge from http://www.rulequest.com/download.html. **[407]**
26. Intelligent Terminals, Ltd. **[407]**
27. See, for example, Jerome H. Friedman, "A Recursive Partitioning Decision Rule for Nonparametric Classification," *IEEE Transactions on Computers*, Vol. 26, No. 4, pp. 404–408, 1977. **[408]**
28. See Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone, *Classification and Regression Trees*, Pacific Grove, CA: Wadsworth, 1984. **[408]**
29. CART 5 is available from Salford Systems. See http://www.salford-systems.com/1112.php. **[408]**
30. J. Ross Quinlan, "Learning Logical Definitions from Relations," *Machine Learning*, Vol. 5, pp. 239–266, 1990. **[408]**
31. The interested reader who is comfortable with logic theory might consult Stephen Muggleton and Luc De Raedt, "Inductive Logic Programming, Theory and Methods," *Journal of Logic Programming*, Vols. 19–20, pp. 629–679, 1994, and Nada Lavrac and Saso Dzeroski, *Inductive Logic Programming: Techniques and Applications*, New York: Ellis Horwood, 1994 (available online at http://www-ai.ijs.si/SasoDzeroski/ILPBook/). See also Claude Sammut, "The Origins of Inductive Logic Programming: A Prehistoric Tale," in Stephen Muggleton (ed.), *Proceedings of the Third International Workshop on Inductive Logic Programming*, pp. 127–147, Bled, Slovenia, 1993. **[408]**
32. See Saso Dzeroski and Nada Lavrac (eds.), *Relational Data Mining*, Berlin: Springer-Verlag, 2001. **[408]**
33. David Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, in James L. McClelland, David E. Rumelhart, and the PDP Research Group (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pp. 318–362, Cambridge, MA: MIT Press, 1986. See also David Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, Vol. 323, Letters, pp. 533–536, October 9, 1986. **[409]**
34. Arthur E. Bryson Jr. and Y. C. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*, Waltham, MA: Blaisdell, 1969. **[409]**
35. Paul Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," Ph.D. thesis, Harvard University, Cambridge, MA, 1974. Laveen Kanal, on the occasion of his acceptance of the 1992 King-Sun Fu award of the International Association for Pattern Recognition (IAPR), recalls a 1975 conversation with Werbos. See Laveen N. Kanal, "On Pattern, Categories, and Alternate Realities," *Pattern Recognition Letters*, Vol. 14, pp. 241–255, 1993. Available online at http://www.lnk.com/prl14.pdf. **[409]**
36. Laveen N. Kanal, "On Pattern, Categories, and Alternate Realities," *Pattern Recognition Letters*, Vol. 14, pp. 241–255, 1993. Available online at http://www.lnk.com/prl14.pdf. **[409]**
37. Terrence J. Sejnowski and Charles R. Rosenberg, "Parallel Networks That Learn to Pronounce English Text," *Complex Systems*, Vol. 1, pp. 145–168, 1987. Available online at http://www.cnl.salk.edu/ParallelNetsPronounce/ParallelNetsPronounce-TJSejnowski.pdf. **[410]**
38. An early paper was Dean Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network," *Advances in Neural Information Processing Systems*, Vol. 1,

pp. 305–313, San Francisco: Morgan Kaufmann Publishers, 1989. Pomerleau's thesis was "Neural Network Perception for Mobile Robot Guidance," Carnegie Mellon University, February 1992. **[411]**

39. Charles Thorpe *et al.*, "Vision and Navigation for the Carnegie Mellon Navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 3, pp. 362–373, May 1988. **[411]**
40. Dean A. Pomerleau, "Neural Network Vision for Robot Driving," in Michael Arbib (ed.), *The Handbook of Brain Theory and Neural Networks*, Cambridge, MA: MIT Press, 1995. A version of this paper is available online at http://www.ri.cmu.edu/pub_files/pub2/pomerleau_dean_1995_1/pomerleau_dean_1995_1.pdf. **[411]**
41. *Ibid*. **[413]**
42. Dean Pomerleau, "RALPH: Rapidly Adapting Lateral Position Handler," *Proceedings of the IEEE Symposium on Intelligent Vehicles*, pp. 506–511, September 1995. Available online at http://www.ri.cmu.edu/pub_files/pub2/pomerleau_dean_1995_2/pomerleau_dean_1995_2.pdf. **[413]**
43. The "No Hands Across America" homepage is at http://www.cs.cmu.edu/afs/cs/usr/tjochem/www/nhaa/nhaa_home_page.html. There are pointers on that site to the trip's journal and photos. For more recent work, see the NavLab's homepage at http://www.ri.cmu.edu/labs/lab_28.html. **[413]**
44. Peter Cheeseman *et al.*, "AutoClass: A Bayesian Classification System," *Proceedings of the Fifth International Conference on Machine Learning*, pp. 54–64, San Francisco: Morgan Kaufmann Publishers, 1988. See also Peter Cheeseman and J. Stutz, "Bayesian Classification (AutoClass): Theory and Results," in Usama M. Fayyad *et al.* (eds.), *Advances in Knowledge Discovery and Data Mining*, Menlo Park, CA: AAAI Press and Cambridge, MA: MIT Press, 1996. Available online at http://ti.arc.nasa.gov/m/project/autoclass/kdd-95.ps. **[414]**
45. http://ti.arc.nasa.gov/project/autoclass/. **[414]**
46. Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*, New York: John Wiley and Sons, Inc., 2001. **[415]**
47. Edward L. Thorndike, *Animal Intelligence*, New York: The Macmillan Co., 1911. **[415]**
48. Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, Section 1.6, Cambridge, MA: MIT Press, 1998; available online at http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html. **[415]**
49. John H. Andreae, "STeLLA: A Scheme for a Learning Machine," *Proceedings of the 2nd IFAC Congress*, 1963. Published in *Automation and Remote Control*, London: Butterworths, 1964. **[415]**
50. Donald Michie, "Experiments on the Mechanisation of Game Learning: 1. Characterization of the Model and its Parameters," *Computer Journal*, Vol. 1, pp. 232–263, 1963. **[415]**
51. Donald Michie and R. Chambers, "BOXES: An Experiment in Adaptive Control," in E. Dale and Donald Michie (eds.), *Machine Intelligence 2*, pp. 137–152, Edinburgh: Oliver and Boyd, 1968. **[415]**
52. A. Harry Klopf, *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*, Washington, DC: Hemisphere, 1982. **[415]**
53. Christopher J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. thesis, Cambridge University, Cambridge, England, 1989. **[417]**
54. Andrew Moore and Christopher G. Atkeson, "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time," *Machine Learning*, Vol. 13, October 1993. Online version available at http://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1993_1/moore_andrew_1993_1.pdf. **[419]**

55. For more information, see, for example, the following: Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 1998 (an html version of the book is available online at http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html.), and Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237–285, 1996. A Web page with lots of pointers to papers and demonstrations is at http://rlai.cs.ualberta.ca/RLAI/rlai.html. **[420]**
56. Gerald Tesauro, "Temporal Difference Learning and TD-GAMMON," *Communications of the ACM*, Vol. 38, No. 3, March 1995. An html version of the paper is available online at http://www.research.ibm.com/massive/tdl.html. **[420]**
57. The quotation is taken from the html version of their book on reinforcement learning at http://www.cs.ualberta.ca/~sutton/book/11/node2.html. **[421]**
58. Pieter Abbeel *et al.*, "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," in Bernhard Scholkopf, John Platt, and Thomas Hofmann (eds.), *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, pp. 1–8, Cambridge, MA: MIT Press, 2007. A pdf version is available at http://www.cs.stanford.edu/~ang/papers/nips06-aerobatichelicopter.pdf. Videos are available at http://www.cs.stanford.edu/group/helicopter See the roll video at http://www.cs.stanford.edu/group/helicopter/video/rolls_080130_web960.mp4. **[421]**
59. Chrisopher H. Donahue and Hyojung Seo, "Attaching Values to Actions: Action and Outcome Encoding in the Primate Caudate Nucleus," *The Journal of Neuroscience*, Vol. 28, No. 18, pp. 4579–4580, April 30, 2008. The authors refer to Sutton and Barto's book as well as to the earlier paper by Wolfram Schultz, Peter Dayan, and P. Read Montague, "A Neural Substrate of Prediction and Reward," *Science*, Vol. 275, No. 5306, pp. 1593–1599, March 14, 1997. **[421]**
60. Leo Breiman, "Bagging Predictors," Department of Statistics Technical Report No. 421, University of California, Berkeley, September 1994. Available online at http://salford-systems.com/doc/BAGGING_PREDICTORS.pdf; and Leo Breiman, "Bagging Predictors," *Machine Learning*, Vol. 24, No. 2, pp. 123–140, 1996. **[422]**
61. Robert E. Schapire, "The Strength of Weak Learnability," *Machine Learning*, Vol. 5, pp. 197–227, 1990. Available online at http://www.cs.princeton.edu/~schapire/papers/strengthofweak.pdf. **[423]**
62. Yoav Freund and Robert E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, Vol. 55, No. 1, pp. 119–139, 1997. Compressed PostScript version available online at http://www.cs.princeton.edu/~schapire/papers/FreundSc95.ps.Z. **[423]**
63. See S. B. Kotsiantis and P. E. Pintelas. "Combining Bagging and Boosting," *International Journal of Computational Intelligence*, Vol. 1, No. 4, pp. 324–333, 2004. Available online at http://www.math.upatras.gr/~esdlab/en/members/kotsiantis/ijci paper kotsiantis.pdf. **[423]**
64. H. Glucksman, "On the Improvement of a Linear Separation by Extending the Adaptive Process with a Stricter Condition," *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 6, pp. 941–944, 1966. **[424]**
65. Nello Cristianini and John Shawe-Taylor, *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*, Cambridge, UK: Cambridge University Press, 2000. **[424]**
66. See http://nips.cc/. **[425]**
67. Donald Michie, D.J. Spiegelhalter, and C.C. Taylor (eds.), *Machine Learning, Neural and Statistical Classification*, Chichester: Ellis Horwood, 1994. The book is now out of print but is available online from http://www.maths.leeds.ac.uk/~charles/statlog/. **[425]**

68. Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, second edition, pp. 752–754, Upper Saddle River, NJ: Prentice Hall, 2003. **[425]**
69. For additional perspective on comparing different algorithms, see David J. Hand, "Classifier Technology and the Illusion of Progress," *Statistical Science*, Vol. 21, No. 1, pp. 1–15, 2006. Available online at http://arxiv.org/pdf/math.ST/0606441. **[425]**