PART 5
Knowledge Acquisition

16
General Concepts in Knowledge Acquisition

The success of knowledge-based systems lies in the quality and extent of the knowledge available to the system. Acquiring and validating a large corpus of consistent, correlated knowledge is not a trivial problem. This has given the acquisition process an especially important role in the design and implementation of these systems. Consequently, effective acquisition methods have become one of the principal challenges for the AI research community.

16.1 INTRODUCTION

The goals in this branch of AI are the discovery and development of efficient, cost effective methods of acquisition. Some important progress has recently been made in this area with the development of sophisticated editors and some impressive machine learning programs. But much work still remains before truly general purpose acquisition is possible. In this chapter, we consider general concepts related to acquisition and learning. We begin with a taxonomy of learning based on definitions of behavioral learning types, assess the difficulty in collecting and assimilating large quantities of well correlated knowledge, describe a general model for learning, and examine different performance measures related to the learning process.

Definitions

Knowledge acquisition is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired. Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task. Therefore, we will think of acquisition as goal oriented creation and refinement of knowledge. We take a broad view of the definition here and include autonomous acquisition, contrary to many workers in the field who regard acquisition solely as the process of knowledge elicitation from experts.

Acquired knowledge may consist of facts, rules, concepts, procedures, heuristics, formulas, relationships, statistics, or other useful information. Sources of this knowledge may include one or more of the following.

Experts in the domain of interest
Textbooks
Technical papers
Databases
Reports
The environment

We will consider machine learning as a specialized form of acquisition. It is any method of autonomous knowledge creation or refinement through the use of computer programs.

Table 16.1 depicts several types of knowledge and possible representation structures which by now should be familiar. In building a knowledge base, it is

TABLE 16.1 TYPES OF KNOWLEDGE AND POSSIBLE STRUCTURES

Type of Knowledge	Examples of Structures
Facts	(snow color white)
Relations	(father_of john bill)
Rules	(if (temperature>200 degrees) (open relief_valve))
Concepts	(forall (x y) (if (and (male x) ((brother_of x) (or (father_of y) (mother_of y))) (uncle x y)]
Procedures, Plans, etc.	. . . .

necessary to create or modify structures such as these for subsequent use by a performance component (like a theorem prover or an inference engine).

To be effective, the newly acquired knowledge should be integrated with existing knowledge in some meaningful way so that nontrivial inferences can be drawn from the resultant body of knowledge. The knowledge should, of course, be accurate, nonredundant, consistent (noncontradictory), and fairly complete in the sense that it is possible to reliably reason about many of the important conclusions for which the system was intended.

16.2 TYPES OF LEARNING

We all learn new knowledge through different methods, depending on the type of material to be learned, the amount of relevant knowledge we already possess, and the environment in which the learning takes place. It should not come as a surprise to learn that many of these same types of learning methods have been extensively studied in AI.

In what follows, it will be helpful to adopt a classification or taxonomy of learning types to serve as a guide in studying or comparing differences among them. One can develop learning taxonomies based on the type of knowledge representation used (predicate calculus, rules, frames), the type of knowledge learned (concepts, game playing, problem solving), or by the area of application (medical diagnosis, scheduling, prediction, and so on). The classification we will use, however, is intuitively more appealing and one which has become popular among machine learning researchers. The classification is independent of the knowledge domain and the representation scheme used. It is based on the type of inference strategy employed or the methods used in the learning process.

The five different learning methods under this taxonomy are

Memorization (rote learning)
Direct instruction (by being told)
Analogy
Induction
Deduction

Learning by memorization is the simplest form of learning. It requires the least amount of inference and is accomplished by simply copying the knowledge in the same form that it will be used directly into the knowledge base. We use this type of learning when we memorize multiplication tables, for example.

A slightly more complex form of learning is by direct instruction. This type of learning requires more inference than rote learning since the knowledge must be transformed into an operational form before being integrated into the knowledge base. We use this type of learning when a teacher presents a number of facts directly to us in a well organized manner.

The third type listed, analogical learning, is the process of learning a new concept or solution through the use of similar known concepts or solutions. We use this type of learning when solving problems on an exam where previously learned examples serve as a guide or when we learn to drive a truck using our knowledge of car driving. We make frequent use of analogical learning. This form of learning requires still more inferring than either of the previous forms, since difficult transformations must be made between the known and unknown situations.

The fourth type of learning is also one that is used frequently by humans. It is a powerful form of learning which, like analogical learning, also requires more inferring than the first two methods. This form of learning requires the use of inductive inference, a form of invalid but useful inference. We use inductive learning when we formulate a general concept after seeing a number of instances or examples of the concept. For example, we learn the concepts of color or sweet taste after experiencing the sensations associated with several examples of colored objects or sweet foods.

The final type of acquisition is deductive learning. It is accomplished through a sequence of deductive inference steps using known facts. From the known facts, new facts or relationships are logically derived. For example, we could learn deductively that Sue is the cousin of Bill, if we have knowledge of Sue and Bill's parents and rules for the cousin relationship. Deductive learning usually requires more inference than the other methods. The inference method used is, of course, a deductive type, which is a valid form of inference.

In addition to the above classification, we will sometimes refer to learning methods as either weak methods or knowledge-rich methods. Weak methods are general purpose methods in which little or no initial knowledge is available. These methods are more mechanical than the classical AI knowledge-rich methods. They often rely on a form of heuristic search in the learning process. Examples of some weak learning methods are given in the next chapter under the names of Learning Automata and Genetic Algorithms. We will be studying these and many of the more knowledge-rich forms of learning in more detail later, particularly various types of inductive learning.

16.3 KNOWLEDGE ACQUISITION IS DIFFICULT

One of the important lessons learned by AI researchers during the 1970s and early 1980s is that knowledge is not easily acquired and maintained. It is a difficult and time-consuming process. Yet expert and other knowledge-based systems require an abundant amount of well correlated knowledge to achieve a satisfactory level of intelligent performance. Typically, tens of person years are often required to build up a knowledge base to an acceptable level of performance. This was certainly true for the early expert systems such as MYCIN, DENDRAL, PROSPECTOR, and XCON. The acquisition effort encountered in building these systems provided the impetus for researchers to search for new efficient methods of acquisition. It helped to revitalize a new interest in general machine learning techniques.

Early expert systems initially had a knowledge base consisting of a few hundred rules. This is equivalent to less than 10^6 bits of knowledge. In contrast, the capacity of a mature human brain has been estimated at some 10^{15} bits of knowledge (Sagan, 1977). If we expect to build expert systems that are highly competent and possess knowledge in more than a single narrow domain, the amount of knowledge required for such knowledge bases will be somewhere between these two extremes, perhaps as much as 10^{10} bits.

If we were able to build such systems at even ten times the rate these early systems were built, it would still require on the order of 10^4 person years. This estimate is based on the assumption that the time required is directly proportional to the size of the knowledge base, a simplified assumption, since the complexity of the knowledge and the interdependencies grow more rapidly with the size of the knowledge base.

Clearly, this rate of acquisition is not acceptable. We must develop better acquisition and learning methods before we can implement such systems within a realistic time frame. Even with the progress made in the past few years through the development of special editors and related tools, more significant breakthroughs are needed before truly large knowledge bases can be assembled and maintained. Because of this, we expect the research interest in knowledge acquisition and machine learning to continue to grow at an accelerated rate for some years in the future.

It has been stated before that a system's performance is strongly dependent on the level and quality of its knowledge, and that "in knowledge lies power." If we accept this adage, we must also agree that the acquisition of knowledge is of paramount importance and, in fact, that "the real power lies in the ability to acquire new knowledge efficiently." To build a machine that can learn and continue to improve its performance has been a long time dream of mankind. The fulfillment of that dream now seems closer than ever before with the modest successes achieved by AI researchers over the past twenty years.

We will consider the complexity problem noted above again from a different point of view when we study the different learning paradigms.

16.4 GENERAL LEARNING MODEL

As noted earlier, learning can be accomplished using a number of different methods. For example, we can learn by memorizing facts, by being told, or by studying examples like problem solutions. Learning requires that new knowledge structures be created from some form of input stimulus. This new knowledge must then be assimilated into a knowledge base and be tested in some way for its utility. Testing means that the knowledge should be used in the performance of some task from which meaningful feedback can be obtained, where the feedback provides some measure of the accuracy and usefulness of the newly acquired knowledge.

A general learning model is depicted in Figure 16.1 where the environment has been included as part of the overall learner system. The environment may be regarded as either a form of nature which produces random stimuli or as a more

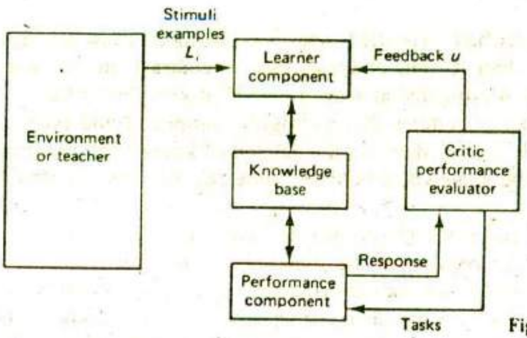


Figure 16.1 General learning model.

organized training source such as a teacher which provides carefully selected training examples for the learner component. The actual form of environment used will depend on the particular learning paradigm. In any case, some representation language must be assumed for communication between the environment and the learner. The language may be the same representation scheme as that used in the knowledge base (such as a form of predicate calculus). When they are chosen to be the same, we say the single representation trick is being used. This usually results in a simpler implementation since it is not necessary to transform between two or more different representations.

For some systems the environment may be a user working at a keyboard. Other systems will use program modules to simulate a particular environment. In even more realistic cases, the system will have real physical sensors which interface with some world environment.

Inputs to the learner component may be physical stimuli of some type or descriptive, symbolic training examples. The information conveyed to the learner component is used to create and modify knowledge structures in the knowledge base. This same knowledge is used by the performance component to carry out some tasks, such as solving a problem, playing a game, or classifying instances of some concept.

When given a task, the performance component produces a response describing its actions in performing the task. The critic module then evaluates this response relative to an optimal response.

Feedback, indicating whether or not the performance was acceptable, is then sent by the critic module to the learner component for its subsequent use in modifying the structures in the knowledge base. If proper learning was accomplished, the system's performance will have improved with the changes made to the knowledge base.

The cycle described above may be repeated a number of times until the performance of the system has reached some acceptable level, until a known learning goal has been reached, or until changes cease to occur in the knowledge base after some chosen number of training examples have been observed.

There are several important factors which influence a system's ability to learn in addition to the form of representation used. They include the types of training provided, the form and extent of any initial background knowledge, the type of feedback provided, and the learning algorithms used (Figure 16.2).

The type of training used in a system can have a strong effect on performance, much the same as it does for humans. Training may consist of randomly selected instances or examples that have been carefully selected and ordered for presentation. The instances may be positive examples of some concept or task being learned, they may be negative, or they may be a mixture of both positive and negative. The instances may be well focused using only relevant information, or they may contain a variety of facts and details including irrelevant data.

Many forms of learning can be characterized as a search through a space of possible hypotheses or solutions (Mitchell, 1982). To make learning more efficient, it is necessary to constrain this search process or reduce the search space. One method of achieving this is through the use of background knowledge which can be used to constrain the search space or exercise control operations which limit the search process. We will see several examples of this in the next three chapters.

Feedback is essential to the learner component since otherwise it would never know if the knowledge structures in the knowledge base were improving or if they were adequate for the performance of the given tasks. The feedback may be a simple yes or no type of evaluation, or it may contain more useful information describing why a particular action was good or bad. Also, the feedback may be completely reliable, providing an accurate assessment of the performance or it may contain noise; that is, the feedback may actually be incorrect some of the time. Intuitively, the feedback must be accurate more than 50% of the time; otherwise the system would never learn. If the feedback is always reliable and carries useful information, the learner should be able to build up a useful corpus of knowledge quickly. On the other hand, if the feedback is noisy or unreliable, the learning process may be very slow and the resultant knowledge incorrect.

Finally, the learning algorithms themselves determine to a large extent how successful a learning system will be. The algorithms control the search to find and build the knowledge structures. We then expect that the algorithms that extract much of the useful information from training examples and take advantage of any background knowledge outperform those that do not. In the following chapters we

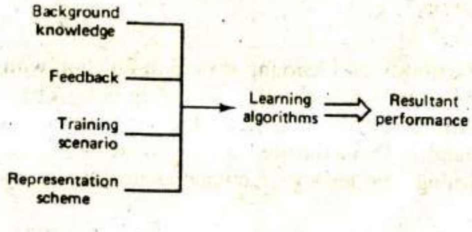


Figure 16.2 Factors affecting learning performance.

will see examples of systems which illustrate many of the above points regarding the effects of different factors on performance.

16.5 PERFORMANCE MEASURES

In the following four chapters, we will be investigating systems based on different learning paradigms and having different architectures. How can we evaluate the performance of a given system or compare the relative performance of two different systems? We could attempt to conduct something like a Turing test on a system. But would this tell us how general or robust the system is or how easy it is to implement? Clearly, such comparisons are possible only when standard performance measures are available. For example, it would be useful to establish the relative efficiencies or speed with which two systems learned a concept or how robust (noise tolerant) a system is under different training scenarios. Although little work has been done to date in this area, we will propose some informal definitions for performance measures in this section. These definitions will at least permit us to establish estimates of some relative performance characteristics among the different learning methods we will be considering.

Generality. One of the most important performance measures for learning methods is the generality or scope of the method. Generality is a measure of the ease with which the method can be adapted to different domains of application. A completely general algorithm is one which is a fixed or self adjusting configuration that can learn or adapt in any environment or application domain. At the other extreme are methods which function in a single domain only. Methods which have some degree of generality will function well in at least a few domains.

Efficiency. The efficiency of a method is a measure of the average time required to construct the target knowledge structures from some specified initial structures. Since this measure is often difficult to determine and is meaningless without some standard comparison time, a relative efficiency index can be used instead. For example, the relative efficiency of a method can be defined as the ratio of the time required for the given method to the time required for a purely random search to find the target structures.

Robustness. Robustness is the ability of a learning system to function with unreliable feedback and with a variety of training examples, including noisy ones. A robust system must be able to build tentative structures which are subject to modification or withdrawal if later found to be inconsistent with statistically sound structures. This is nonmonotonic learning, the analog of nonmonotonic reasoning discussed in Chapter 5.

Efficacy. The efficacy of a system is a measure of the overall power of the system. It is a combination of the factors generality, efficiency, and robustness. We say that system A is more efficacious than system B if system A is more efficient, robust, and general than B.

Ease of implementation. Ease of implementation relates to the complexity of the programs and data structures and the resources required to develop the given learning system. Lacking good complexity metrics, this measure will often be somewhat subjective.

Other performance terms that are specific to different paradigms will be introduced as needed.

16.6 SUMMARY

Knowledge acquisition is the purposeful addition or refinement of knowledge structures to a knowledge base for use by knowledge-based systems. Machine learning is the autonomous acquisition of knowledge through the use of computer programs. The acquired knowledge may consist of facts, concepts, rules, relations, plans, and procedures, and the source of the knowledge may be one or more of the following: data bases, textbooks, domain experts, reports, or the environment.

A useful taxonomy for learning is one that is based on the behavioral strategy employed in the learning process, strategies like rote (memorization), being told, analogy, induction, or deduction. Rote learning requires little inference, while the other methods require increasingly greater amounts.

An important lesson learned by expert systems researchers is that knowledge acquisition is difficult, often requiring tens of person years to assemble several hundred rules. This problem helped to revive active research in more autonomous forms of acquisition or machine learning.

Any model of learning should include components which support the basic learner component. These include a teacher or environment, a knowledge base, a performance component which uses the knowledge, and a critic or performance evaluation unit which provides feedback to the learner about the performance.

Factors which affect the performance of a learner system include (1) the representation scheme used, (2) the training scenario, (3) the type of feedback, (4) background knowledge, and (5) the learning algorithm.

Performance measures provide a means of comparing different performance characteristics of learning algorithms (systems). Some of the more important performance measures are generality, efficiency, robustness, efficacy, and ease of implementation. Lacking formal metrics for some measures will require that subjective estimates be used instead.

EXERCISES

16.1. Give a detailed definition of knowledge acquisition complete with an example.

16.2. Give an example for each of the following types of knowledge: (a) a fact, (b) a rule, (c) a concept, (d) a procedure, (e) a heuristic, (f) a relationship.

16.3. How is machine learning distinguished from general knowledge acquisition?

16.4. The taxonomy of learning methods given in this chapter is based on the type of behavior or inference strategy used. Give another taxonomy for learning methods and illustrate with some examples.

16.5. Describe the role of each component of a general learning model and why it is needed for the learning process.

16.6. Explain why a learning component should have scope.

16.7. What is the difference between efficiency and efficacy with regard to the performance of a learner system?

16.8. Review the knowledge acquisition section in Chapter 15 and explain why elicitation of knowledge from experts is so difficult.

16.9. Consult a good dictionary and describe the difference between induction and deduction. Give examples of both.

16.10. Explain why inductive learning should require more inference than learning by being told (instruction).

16.11. Try to determine whether inductive learning requires more inference than analogical learning. Give reasons for your conclusion. Which of the two types of learning, in general, would be more reliable in the sense that the knowledge learned is logically valid?

16.12. Give examples of each of the different types of learning as they relate to learning you have recently experienced.

16.13. Give an estimate of the number of bits required to store knowledge for 500 if . . . then rules, where each rule has five conjunctive terms or conditions in the antecedent. Can you estimate what fraction of your total knowledge is represented in some set of 500 rules?

16.14. Try to give a quantitative measure for knowledge. Explain why your measure is or is not reasonable. Give examples to support your arguments.

16.15. Explain why robustness in a learner is important in real world environments.

17 Early Work in Machine Learning

17.1 INTRODUCTION

Attempts to develop autonomous learning systems began in the 1950s while cybernetics was still an active area of research. These early designs were self-adapting systems which modified their own structures in an attempt to produce an optimal response to some input stimuli. Although several different approaches were pursued during this period, we will consider only four of the more representative designs. One approach was believed to be an approximate model of a small network of neurons.

A second approach was initially based on a form of rote learning. It was later modified to learn by adaptive parameter adjustment. The third approach used self-adapting stochastic automata models, while the fourth approach was modeled after survival of the fittest through population genetics.

In the remainder of this chapter, we will examine examples of the four methods and reserve our descriptions of more classical AI learning approaches for Chapters 18, 19, and 20. The systems we use as examples here are famous ones that have received much attention in the literature. They are Rosenblatt's perceptrons, Samuel's checkers playing system, Learning Automata, and Genetic Algorithms.

17.2 PERCEPTRONS

Perceptrons are pattern recognition or classification devices that are crude approximations of neural networks. They make decisions about patterns by summing up evidence obtained from many small sources. They can be taught to recognize one or more classes of objects through the use of stimuli in the form of labeled training examples.

A simplified perceptron system is illustrated in Figure 17.1. The inputs to the system are through an array of sensors such as a rectangular grid of light sensitive pixels. These sensors are randomly connected in groups to associative threshold units (ATU) where the sensor outputs are combined and added together. If the combined outputs to an ATU exceed some fixed threshold, the ATU unit executes and produces a binary output.

The outputs from the ATU are each multiplied by adjustable parameters or weights w_i $(i = 1, 2, \ldots, k)$ and the results added together in a terminal comparator unit. If the input to the comparator exceeds a given threshold level T, the perceptron produces a positive response of 1 (yes) corresponding to a sample classification of class-1. Otherwise, the output is 0 (no) corresponding to an object classification of non-class-1

All components of the system are fixed except the weights w_i which are adjusted through a punishment-reward process described below. This learning process contin-

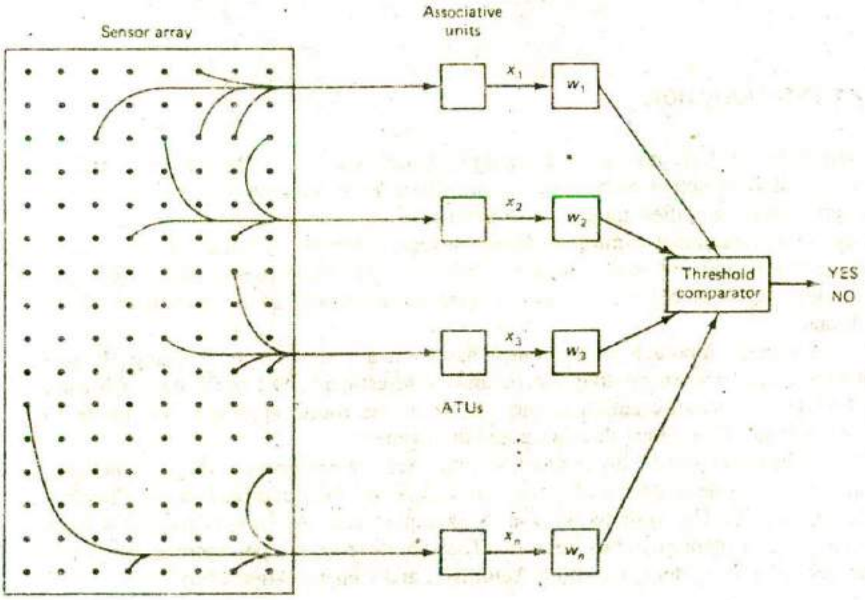


Figure 17.1 A simple perceptron.

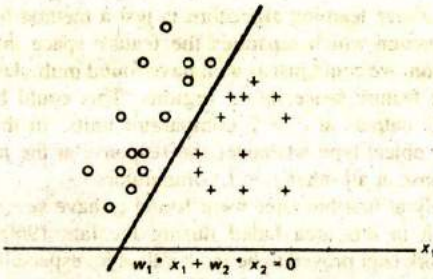


Figure 17.2 Geometrical illustration of separable space.

ues until optimal values of w_i are found at which time the system will have learned the proper classification of objects for the two different classes.

The light sensors produce an output voltage that is proportional to the light intensity striking them. This output is a measure of what is known in pattern recognition as the object representation space parameters. The outputs from the ATUs which combine several of the sensor outputs (x_i) are known as feature value measurements. These feature values are each multiplied by the weights w_i and the results summed in the comparator to give the vector product $\mathbf{r} = \mathbf{w} * \mathbf{x} = \Sigma_i w_i x_i$. When enough of the feature values are present and the weight vector is near optimal, the threshold will be exceeded and a positive classification will result.

Finding the optimal weight vector value **w** is equivalent to finding a separating hyperplane in k-dimensional space. If there is some linear function of the x_i for which objects in class-1 produce an output greater than T and non-class-1 objects produce an output less than T, the space of objects is said to be linearly separable (see Chapter 13 for example). Spaces which are linearly separable can be partitioned into two or more disjointed regions which divide objects based on their feature vector values as illustrated in Figure 17.2. It has been shown (Minsky and Papert, 1969) that an optimal **w** can always be found with a finite number of training examples if the space is linearly separable.
One of the simplest algorithms for finding an optimum **w** (**w***) is based on the following perceptron learning algorithm. Given training objects from two distinct classes, class-1 and class-2

1. Choose an arbitrary initial value for **w**
2. After the m^{th} training step set

$$\mathbf{w}_{m+1} = \mathbf{w}_m + d * \mathbf{x}_m$$

where

$$d = +1 \text{ if } r = \mathbf{w} * \mathbf{x} < 0 \quad \text{and} \quad \mathbf{x}_m \text{ is type class-1}$$

$$d = -1 \text{ if } r > 0 \quad \text{and} \quad \mathbf{x}_m \text{ is type class-2}$$

Otherwise set $\mathbf{w}_{m+1} = \mathbf{w}_m\ (d = 0)$

3. When $\mathbf{w}_m = \mathbf{w}_m +_j$. (for all $j >= 1$) stop, the optimum **w*** has been found.

It should be recognized that the above learning algorithm is just a method for finding a linear two-class decision function which separates the feature space into two regions. For a generalized perceptron, we could just as well have found multiclass decision functions which separate the feature space into c regions. This could be done by terminating each of the ATU outputs at $c > 2$ comparator units. In this case, class j would be selected as the object type whenever the response at the jth comparator was greater than the response at all other $j - 1$ comparators.

Perceptrons were studied intensely at first but later were found to have severe limitations. Therefore, active research in this area faded during the late 1960s. However, the findings related to this work later proved to be most valuable, especially in the area of pattern recognition. More recently, there has been renewed interest in similar architectures which attempt to model neural networks (Chapter 15). This is partly due to a better understanding of the brain and significant advances realized in network dynamics as well as in hardware over the past ten years. These advances have made it possible to more closely model large networks of neurons.

17.3 CHECKERS PLAYING EXAMPLE

During the 1950s and 1960s Samuel (1959, 1967) developed a program which could learn to play checkers at a master's level. This system remembered thousands of board states and their estimated values. They provided the means to determine the best move to make at any point in the game.

Samuel's system learns while playing the game of checkers, either with a human opponent or with a copy of itself. At each state of the game, the program checks to see if it has remembered a best-move value for that state. If not, the program explores ahead three moves (it determines all of its possible moves; for each of these, it finds all of its opponent's moves; and for each of those, it determines all of its next possible moves). The program then computes an advantage or win-value estimate of all the ending board states. These values determine the best move for the system from the current state. The current board state and its corresponding value are stored using an indexed address scheme for subsequent recall.

The best move for each state is the move value of the largest of the minimums, based on the theory of two-person zero-sum games. This move will always be the best choice (for the next three moves) against an intelligent adversary.

As an example of the look-ahead process, a simple two move sequence is illustrated in Figure 17.3 in the form of a tree. At board state K, the program looks ahead two moves and computes the value of each possible resultant board state. It then works backward by first finding the minimum board values at state $K + 2$ in each group of moves made from state $K + 1$ (minimums = 4, 3, and 2).

These minimums correspond to the moves the opponent would make from each position when at state $K + 1$. The program then chooses the maximum of these minimums as the best (minimax) move it can make from the present board

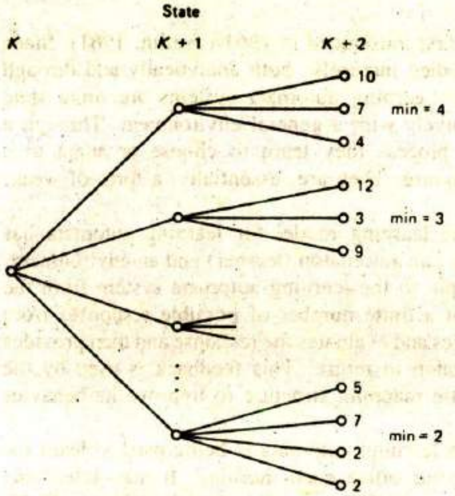


Figure 17.3 A two-move look-ahead sequence.

state K (maximum = 4). By looking ahead three moves, the system can be assured it can do no worse than this minimax value. The board state and the corresponding minimax value for a three-move-ahead sequence are stored in Samuel's system. These values are then available for subsequent use when the same state is encountered during a new game.

The look ahead search process could be extended beyond three moves; however, the combinatorial explosion that results makes this infeasible. But, when many board states have been learned, it is likely that any given state will already have look-ahead values for three moves, and some of those moves will in turn have look-ahead values stored. Consequently, as more and more values are stored, look-ahead values for six, nine, or even more moves may be prerecorded for rapid use. Thus, when the system has played many games and recorded thousands of moves, its ability to look ahead many moves and to show improved performance is greatly increased.

The value of a board state is estimated by computing a linear function similar to the perceptron linear decision function. In this case, however, Samuel selected some 16 board features from a larger set of feature parameters. The features were typically checkers concepts such as piece advantage, the number of kings and single piece units, and the location of pieces. In the original system, the weighting parameters were fixed. In subsequent experiments, however, the parameters were adjusted as part of the learning process much like the weighting parameters were in the perceptrons.

17.4 LEARNING AUTOMATA

The theory of learning automata was first introduced in 1961 (Tsetlin, 1961). Since that time these systems have been studied intensely, both analytically and through simulations (Lakshmivarahan, 1981). Learning automata systems are finite state adaptive systems which interact iteratively with a general environment. Through a probabilistic trial-and-error response process they learn to choose or adapt to a behavior which produces the best response. They are, essentially, a form of weak, inductive learners.

In Figure 17.4, we see that the learning model for learning automata has been simplified to just two components, an automaton (learner) and an environment. The learning cycle begins with an input to the learning automata system from the environment. This input elicits one of a finite number of possible responses from the automaton. The environment receives and evaluates the response and then provides some form of feedback to the automaton in return. This feedback is used by the automaton to alter its stimulus-response mapping structure to improve its behavior in a more favorable way.

As a simple example, suppose a learning automata is being used to learn the best temperature control setting for your office each morning. It may select any one of ten temperature range settings at the beginning of each day (Figure 17.5). Without any prior knowledge of your temperature preferences, the automaton randomly selects a first setting using the probability vector corresponding to the temperature settings.

Since the probability values are uniformly distributed, any one of the settings will be selected with equal likelihood. After the selected temperature has stabilized, the environment may respond with a simple good-bad feedback response. If the response is good, the automata will modify its probability vector by rewarding the probability corresponding to the good setting with a positive increment and reducing all other probabilities proportionately to maintain the sum equal to 1. If the response is bad, the automaton will penalize the selected setting by reducing the probability corresponding to the bad setting and increasing all other values proportionately. This process is repeated each day until the good selections have high probability values and all bad choices have values near zero. Thereafter, the system will always choose the good settings. If, at some point, in the future your temperature preferences change, the automaton can easily readapt.

Learning automata have been generalized and studied in various ways. One

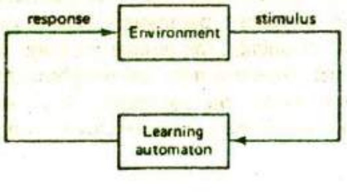


Figure 17.4 Learning automaton model.

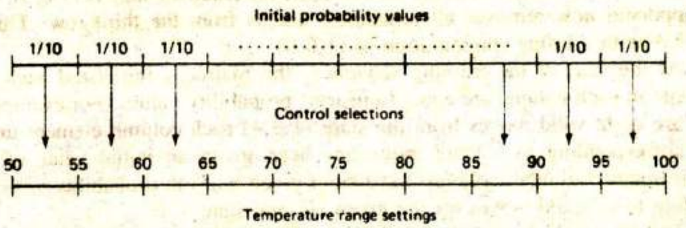


Figure 17.5 Temperature control model.

such generalization has been given the special name of collective learning automata (CLA). CLAs are standard learning automata systems except that feedback is not provided to the automaton after each response. In this case, several collective stimulus-response actions occur before feedback is passed to the automaton. It has been argued (Bock, 1976) that this type of learning more closely resembles that of human beings in that we usually perform a number or group of primitive actions before receiving feedback on the performance of such actions, such as solving a complete problem on a test or parking a car. We illustrate the operation of CLAs with an example of learning to play the game of Nim in an optimal way.

Nim is a two-person zero-sum game in which the players alternate in removing tokens from an array which initially has nine tokens. The tokens are arranged into three rows with one token in the first row, three in the second row, and five in the third row (Figure 17.6).

The first player must remove at least one token but not more than all the tokens in any single row. Tokens can only be removed from a single row during each player's move. The second player responds by removing one or more tokens remaining in any row. Players alternate in this way until all tokens have been removed; the loser is the player forced to remove the last token.

We will use the triple (n_1,n_2,n_3) to represent the states of the game at a given time where n_1, n_2, and n_3 are the numbers of tokens in rows 1, 2, and 3, respectively. We will also use a matrix to determine the moves made by the CLA for any given state. The matrix of Figure 17.7 has heading columns which correspond to the state of the game when it is the CLA's turn to move, and row headings which correspond to the new game state after the CLA has completed a move. Fractional entries in the matrix are transition probabilities used by the CLA to execute each of its moves. Asterisks in the matrix represent invalid moves.

Beginning with the initial state (1,3,5), suppose the CLA's opponent removes two tokens from the third row resulting in the new state (1,3,3). If the CLA then

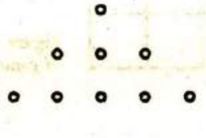


Figure 17.6 Nim initial configuration.

removes all three tokens from the second row, the resultant state is (1,0,3). Suppose the opponent now removes all remaining tokens from the third row. This leaves the CLA with a losing configuration of (1,0,0).

At the start of the learning sequence, the matrix is initialized such that the elements in each column are equal (uniform) probability values. For example, since there are eight valid moves from the state (1,3,4) each column element under this state corresponding to a valid move has been given an initial value of $\frac{1}{8}$. In a similar manner all other columns have been given uniform probability values corresponding to all valid moves for the given column state.

The CLA selects moves probabilistically using the probability values in each column. So, for example, if the CLA had the first move, any row intersecting with the first column not containing an asterisk would be chosen with probability $\frac{1}{9}$. This choice then determines the new game state from which the opponent must select a move. The opponent might have a similar matrix to record game states and choose moves. A complete game is played before the CLA is given any feedback, at which time it is informed whether or not its responses were good or bad. This is the collective feature of the CLA.

If the CLA wins a game, all moves made by the CLA during that game are rewarded by increasing the probability value in each column corresponding to the winning move. All nonwinning probabilities in those columns are reduced equally to keep the sum in each column equal to 1. If the CLA loses a game, the moves leading to that loss are penalized by reducing the probability values corresponding to each losing move. All other probabilities in the columns having a losing move are increased equally to keep the column totals equal to 1.

After a number of games have been played by the CLA, the matrix elements

	Current state						
	135	134	133	132	. . .	125	. . .
135	•	•	•	•	. . .	•	
134	1/9	•	•	•	. . .	•	. . .
133	1/9	1/8	•	•	. . .	•	
132	1/9	1/8	1/7	•	. . .	•	. . .
⋮	⋮	⋮	⋮	⋮			
124	•	1/8	•	•	. . .	1/8	. . .
⋮	⋮	⋮					

Figure 17.7 CLA internal representation of game states.

which correspond to repeated wins will increase toward one, while all other elements in the column will decrease toward zero. Consequently, the CLA will choose the winning moves more frequently and thereby improve its performance.

Simulated games between a CLA and various types of opponents have been performed and the results plotted (Bock, 1985). It was shown, for example, that two CLAs playing against each other required about 300 games before each learned to play optimally. Note, however, that convergence to optimality can be accomplished with fewer games if the opponent always plays optimally (or poorly), since, in such a case, the CLA will repeatedly lose (win) and quickly reduce (increase) the losing (winning) move elements to zero (one). It is also possible to speed up the learning process through the use of other techniques such as learned heuristics.

Learning systems based on the learning automaton or CLA paradigm are fairly general for applications in which a suitable state representation scheme can be found. They are also quite robust learners. In fact, it has been shown that an LA will converge to an optimal distribution under fairly general conditions if the feedback is accurate with probability greater than 0.5 (Narendra and Thathachar, 1974). Of course, the rate of convergence is strongly dependent on the reliability of the feedback.

Learning automata are not very efficient learners as was noted in the game playing example above. They are, however, relatively easy to implement, provided the number of states is not too large. When the number of states becomes large, the amount of storage and the computation required to update the transition matrix becomes excessive.

Potential applications for learning automata include adaptive telephone routing and control. Such applications have been studied using simulation programs (Narendra et al., 1977). Although they have been given favorable recommendations, few if any actual systems have been implemented, however.

17.5 GENETIC ALGORITHMS

Genetic algorithm learning methods are based on models of natural adaptation and evolution. These learning systems improve their performance through processes which model population genetics and survival of the fittest. They have been studied since the early 1960s (Holland, 1962, 1975).

In the field of genetics, a population is subjected to an environment which places demands on the members. The members which adapt well are selected for mating and reproduction. The offspring of these better performers inherit genetic traits from both their parents. Members of this second generation of offspring which also adapt well are then selected for mating and reproduction and the evolutionary cycle continues. Poor performers die off without leaving offspring. Good performers produce good offspring and they, in turn, perform well. After some number of generations, the resultant population will have adapted optimally or at least very well to the environment.

Genetic algorithm systems start with a fixed size population of data structures which are used to perform some given tasks. After requiring the structures to execute the specified tasks some number of times, the structures are rated on their performance, and a new generation of data structures is then created. The new generation is created by mating the higher performing structures to produce offspring. These offspring and their parents are then retained for the next generation while the poorer performing structures are discarded. The basic cycle is illustrated in Figure 17.8.

Mutations are also performed on the best performing structures to insure that the full space of possible structures is reachable. This process is repeated for a number of generations until the resultant population consists of only the highest performing structures.

Data structures which make up the population can represent rules or any other suitable types of knowledge structure. To illustrate the genetic aspects of the problem, assume for simplicity that the population of structures are fixed-length binary strings such as the eight bit string 11010001. An initial population of these eight-bit strings would be generated randomly or with the use of heuristics at time zero. These strings, which might be simple condition and action rules, would then be assigned some tasks to perform (like predicting the weather based on certain physical and geographic conditions or diagnosing a fault in a piece of equipment).

After multiple attempts at executing the tasks, each of the participating structures would be rated and tagged with a utility value *u* commensurate with its performance. The next population would then be generated using the higher performing structures as parents and the process would be repeated with the newly produced generation. After many generations the remaining population structures should perform the desired tasks well.

Mating between two strings is accomplished with the *crossover* operation which randomly selects a bit position in the eight-bit string and concatenates the head of

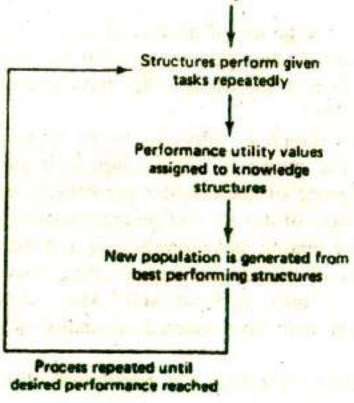

Generate initial population
S_0
Structures perform given tasks repeatedly
Performance utility values assigned to knowledge structures
New population is generated from best performing structures
Process repeated until desired performance reached

Figure 17.8 Genetic algorithm.

one parent to the tail of the second parent to produce the offspring. Suppose the two parents are designated as xxxxxxxx and yyyyyyyy respectively, and suppose the third bit position has been selected as the crossover point (at the position of the colon in the structure xxx:xxxxx). After the crossover operation is applied, two offspring are then generated, namely xxxyyyyy and yyyxxxxx. Such offspring and their parents are then used to make up the next generation of structures.

A second genetic operation often used is called *inversion*. Inversion is a transformation applied to a single string. A bit position is selected at random, and when applied to a structure, the inversion operation concatenates the tail of the string to the head of the same string. Thus, if the sixth position were selected ($x_1x_2x_3x_4x_5x_6{:}x_7x_8$), the inverted string would be $x_7x_8x_1x_2x_3x_4x_5x_6$.

A third operator, *mutation*, is used to insure that all locations of the rule space are reachable, that every potential rule in the rule space is available for evaluation. This insures that the selection process does not get caught in a local minimum. For example, it may happen that use of the crossover and inversion operators will only produce a set of structures that are better than all local neighbors but not optimal in a global sense. This can happen since crossover and inversion may not be able to produce some undiscovered structures. The mutation operator can overcome this by simply selecting any bit position in a string at random and changing it. This operator is typically used only infrequently to prevent random wandering in the search space.

The genetic paradigm is best understood through an example. To illustrate similarities between the learning automaton paradigm and the genetic paradigm we use the same learning task of the previous section, namely learning to play the game of nim optimally. We use a slightly different representation scheme here since we want a population of structures that are easily transformed. To do this, we let each member of the population consist of a pair of triplets augmented with a utility value u, $((n_1,n_2,n_3)\ (m_1,m_2,m_3)u)$, where the first pair is the game state presented to the genetic algorithm system prior to its move, and the second triple is the state after the move. The u values represent the worth or current utility of the structure at any given time.

Before the game begins, the genetic system randomly generates an initial population of K triple-pair members. The population size K is one of the important parameters that must be selected. Here, we simply assume it is about 25 or 30, which should be more than the number of moves needed for any optimal play. All members are assigned an initial utility value of 0. The learning process then proceeds as follows.

1. The environment presents a valid triple to the genetic system.
2. The genetic system searches for all population triple pairs which have a first triple that matches the input triple. From those that match, the first one found having the highest utility value u is selected and the second triple is returned as the new game state. If no match is found, the genetic system randomly generates a triple which represents a valid move, returns this as the new state,

and stores the triple pair, the input, and newly generated triple as an addition to the population.

3. The above two steps are repeated until the game is terminated, in which case the genetic system is informed whether a win or loss occurred. If the system wins, each of the participating member moves has its utility value increased. If the system loses, each participating member has its utility value decreased.
4. The above steps are repeated until a fixed number of games have been played. At this time a new generation is created.

The new generation is created from the old population by first selecting a fraction (say one half) of the members having the highest utility values. From these, offspring are obtained by application of appropriate genetic operators.

The three operators, crossover, inversion, and mutation, randomly modify the parent moves to give new offspring move sequences. (The best choice of genetic operators to apply in this example is left as an exercise). Each offspring inherits a utility value from one of the parents. Population members having low utility values are discarded to keep the population size fixed.

This whole process is repeated until the genetic system has learned all the optimal moves. This can be determined when the parent population ceases to change or when the genetic system repeatedly wins.

The similarity between the learning automaton and genetic paradigms should be apparent from this example. Both rely on random move sequences, and the better moves are rewarded while the poorer ones are penalized.

17.6 INTELLIGENT EDITORS

In the previous chapter, we considered some of the difficulties involved in acquiring and assembling a large corpus of well-correlated knowledge. Expert systems sometimes require hundreds or even thousands of rules to reach acceptable levels of performance. To help alleviate this problem, the development of special editors such as TEIRESIAS (Davis and Lenat, 1982) were initiated during the mid 1970s. Since that time a number of commercial editors have been developed. These intelligent editors have made it possible to build expert systems without strong reliance on knowledge engineers.[1]

An intelligent editor acts as an interface between a domain expert and an expert system. They permit a domain expert to interact directly with the system without the need for an intermediary to code the knowledge (Figure 17.9). The expert carries on a dialog with the editor in a restricted subset of English which includes a domain-specific vocabulary. The editor has direct access to the knowledge in the expert system and knows the structure of that knowledge. Through the editor,

[1] For additional details related to the overall process of eliciting, coding, organizing, and refining knowledge from domain experts, see Chapter 15 which examines expert system architectures and building tools.

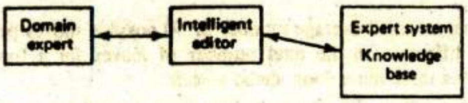


Figure 17.9 Acquisition using an intelligent editor.

an expert can create, modify, and delete rules without a knowledge of the internal structure of the rules.

The editor assists the expert in building and refining a knowledge base by recalling rules related to some specific topic, and reviewing and modifying the rules, if necessary, to better fit the expert's meaning and intent. Through the editor, the expert can query the expert system for conclusions when given certain facts. If the expert is unhappy with the results, a trace can be obtained of the steps followed in the inference process. When faulty or deficit knowledge is found, the problem can then be corrected.

Some editors have the ability to suggest reasonable alternatives and to prompt the expert for clarifications when required. Some editors also have the ability to make validity checks on newly entered knowledge and detect when inconsistencies occur. More recently, a few commercial editors have incorporated features which permit rules to be induced directly from examples of problem solutions. These editors have greatly simplified the acquisition process, but they still require much effort on the part of domain experts.

17.7 SUMMARY

We have examined examples of early work done in machine learning including perceptrons which learn through parameter adjustment, by looking at Samuel's checkers playing system which learns through a form of rote learning as well as parameter adjustment. We then looked at learning automata which uses a reward and punishment process by modifying their state probability transformation mapping structures until optimal performance has been achieved. Genetic algorithm systems learn through a form of mutation and genetic inheritance. Higher performing knowledge structures are mated and give birth to offspring which possess many of their parents' traits. Generations of structures are created until an acceptable level of performance has been reached. Finally, we briefly discussed semiautonomous learning systems, the intelligent editors. These systems permit a domain expert to interact directly in building and refining a knowledge base without strong support from a knowledge engineer.

EXERCISES

17.1. Given a simple perceptron with a 3-x-3 input sensor array, compute six learning cycles to show how the weights w_i change during the learning process. Assign random weights to the initial w_i values.

17.2. For the game of checkers with an assumed average number of 50 possible moves per board position, determine the difference in the total number of moves for a four move look-ahead as compared to a three move look-ahead system.

17.3. Design a learning automaton that selects TV channels based on day of week and time of day (three evening hours only) for some family you are familiar with.

17.4. Write a computer program to simulate the learning automaton of the previous problem. Determine the number of training examples required for the system to converge to the optimal values.

17.5. Describe how a learning automaton could be developed to learn how to play the game of tic-tac-toe optimally. Is this a CLA or a simple learning automaton system?

17.6. Describe the similarities and differences between learning automata and genetic algorithms. Which learner would be best at finding optimal solutions to nonlinear functions? Give reasons to support your answer.

17.7. Explain the difference of the genetic operators inversion, crossover, and mutation. Which operator do you think is most effective in finding the optimal population in the least time?

17.8. Explain why some editors can be distinguished as "intelligent."

17.9. Read an article on TEIRESIUS and make a list of all the intelligent functions it performs that differ from so called nonintelligent editors.

18 Learning by Induction

18.1 INTRODUCTION

Consider playing the following game. I will choose some concept which will remain fixed throughout the game. You will then be given clues to the concept in the form of simple descriptions. After each clue, you must attempt to guess the concept I have chosen. I will continue with the clues until you are sure you have made the right choice.

Clue 1. A diamond ring.

For your first guess did you choose the concept beautiful? If so you are wrong.

Clue 2. Dinner at Maxime's restaurant.

What do clues 1 and 2 have in common? Perhaps clue 3 will be enough to reveal the commonality you need to make a correct choice.

Clue 3. A Mercedes-Benz automobile.

You must have discovered the concept by now! It is an expensive or luxury item.

The above illustrates the process we use in inductive learning, namely, inductive inference, an invalid, but useful form of inference.

In this chapter we continue the study of machine learning, but in a more focused manner. Here, we study a single learning method, learning by inductive inference.

Inductive learning is the process of acquiring generalized knowledge from examples or instances of some class. This form of learning is accomplished through inductive inference, the process of reasoning from a part to a whole, from particular instances to generalizations, or from the individual to the universal. It is a powerful form of learning which we humans do almost effortlessly. Even though it is not a valid form of inference, it appears to work well much of the time. Because of its importance, we have devoted two complete chapters to the subject.

18.2 BASIC CONCEPTS

When we conclude that October weather is always pleasant in El Paso after having observed the weather there for a few seasons, or when we claim that all swans are white after seeing only a small number of white swans, or when we conclude that all Scots are tough negotiators after conducting business with only a few, we are learning by induction. Our conclusions may not always be valid, however. For example, there are also black Australian swans and some weather records show that October weather in El Paso was inclement. Even so, these conclusions or rules are useful. They are correct much or most of the time, and they allow us to adjust our behavior and formulate important decisions with little cognitive effort.

One can only marvel at our ability to formulate a general rule for a whole class of objects, finite or not, after having observed only a few examples. How is it we are able to make this large inductive leap and arrive at an accurate conclusion so easily? For example, how is it that a first time traveler to France will conclude that all French people speak French after having spoken only to one Frenchman named Henri? At the same time, our traveler would not incorrectly conclude that all Frenchmen are named Henri!

Examples like this emphasize the fact that inductive learning is much more than undirected search for general hypotheses. Indeed, it should be clear that inductive generalization and rule formulation are performed in some context and with a purpose. They are performed to satisfy some objectives and, therefore, are guided by related background or other world knowledge. If this were not so, our generalizations would be on shaky ground and our class descriptions might be no more than a complete listing of all the examples we had observed.

The inductive process can be described symbolically through the use of predicates P and Q. If we observe the repeated occurrence of events $P(a_1)$, $P(a_2)$, . . . , $P(a_k)$, we generalize by inductively concluding that $\forall x\ P(x)$, i.e. if (canary_1 color yellow), (canary_2 color yellow), . . . , (canary_*k* color yellow) then (forall *x* (if canary *x*)(*x* color yellow)). More generally, when we observe the implications

$$P(a_1) \rightarrow Q(b_1)$$
$$P(a_2) \rightarrow Q(b_2)$$
$$\vdots$$
$$P(a_k) \rightarrow Q(b_k)$$

we generalize by concluding $\forall xy\ P(x) \rightarrow Q(y)$.

In forming a generalized hypothesis about a complete (possibly infinite) class after observing only a fraction of the members, we are making an uncertain conclusion. Even so, we have all found it to be a most essential form of learning.

Our reliance on and proven success with inductive learning has motivated much research in this area. Numerous systems have been developed to investigate different forms of such learning referred to as learning by observation, learning by discovery, supervised learning, learning from examples, and unsupervised learning. We have already seen two examples of weak inductive learning paradigms in the previous chapter: learning automata and genetic algorithms. In this chapter we will see different kinds of examples which use the more traditional AI architecture.

In the remainder of this and the following chapter, we will study the inductive learning process and look at several traditional AI learning systems based on inductive learning paradigms. We begin in the next two sections with definitions of important terms and descriptions of some essential tools used in the learning systems which follow.

18.3 SOME DEFINITIONS

In this section we introduce some important terms and concepts related to inductive learning. Many of the terms we define will have significance in other areas as well.

Our model of learning is the model described in Section 16.4. As you may recall, our learner component is presented with training examples from a teacher or from the environment. The examples may be positive instances only or both positive and negative. They may be selected, well-organized examples or they may be presented in a haphazard manner and contain much irrelevant information Finally, the examples may be correctly labeled as positive or negative instances of some concept, they may be unlabeled, or they may even contain erroneous labels. Whatever the scenario, we must be careful to describe it appropriately.

Given (1) the observations, (2) certain background domain knowledge, and (3) goals or other preference criteria, the task of the learner is to find an inductive assertion or target concept that implies all of the observed examples and is consistent with both the background knowledge and goals. We formalize the above ideas with the following definitions (Hunt et al., 1966, and Rendell, 1985).

Definitions

Object. Any entity, physical or abstract, which can be described by a set of attribute (feature) values and attribute relations is an object. We will refer to an object either by its name o_i or by an appropriate representation such as the vector of attribute values $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Clearly, the x_i can be very primitive attributes or higher level, more abstract ones. The choice will depend on the use being made of the representations. For example, a car may be described with primitives as an entity made from steel, glass, rubber, and its other component materials or, at a higher level of abstraction, as an object used for the transportation of passengers at speeds ranging up to 250 miles per hour.

Class. Given some universe of objects U, a class is a subset of U. For example, given the universe of four-legged animals, one class is the subset horses.

Concept. This is a description of some class (set) or rule which partitions the universe of objects U into two sets, the set of objects that satsisfy the rule and those that do not. Thus, the concept of horse is a description or rule which asserts the set of all horses and excludes all nonhorses.

Hypothesis. A hypothesis H is an assertion about some objects in the universe. It is a candidate or tentative concept which partitions the universe of objects. One such hypothesis related to the concept of horse is the class of four-legged animals with a tail. This is a candidate (albeit incomplete) for the concept horse.

Target concept. The target is one concept which correctly classifies all objects in the universe.

Positive instances. These are example objects which belong to the target concept.

Negative instances. These are examples opposite to the target concept.

Consistent classification rule. This is a rule that is true for all positive instances and false for all negative instances.

Induction. Induction is the process of class formation. Here we are more interested in the formation of classes which are goal-oriented; therefore, we will define induction as purposeful class formation. To illustrate, from our earlier example, we concluded that all Scotsmen are tough negotiators. Forming this concept can be helpful when dealing with any Scot. The goal or purpose here is in the simplification of our decisions when found in such situations.

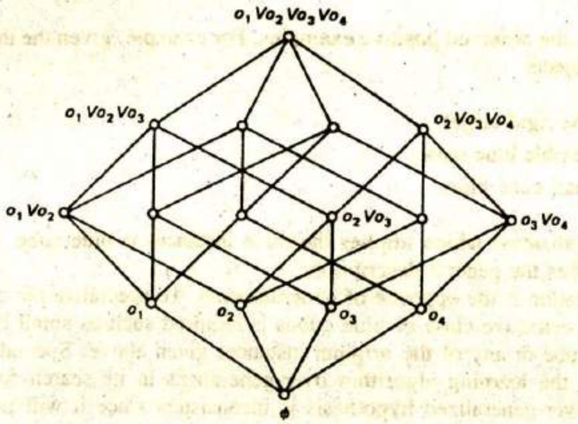


Figure 18.1 Lattice of object classes. Each node represents a disjunction of objects from a universe of four objects.

Selective induction. In this form of induction class descriptions are formed using only the attributes and relations which appear in the positive instances.

Constructive induction. This form of induction creates new descriptors not found in any of the instances.

Expedient induction. This is the application of efficient, efficacious inductive learning methods which have some scope, methods which span more than a single domain. The combined performance in efficiency, efficacy, and scope has been termed the inductive power of a system (Rendell, 1985).

In this chapter we are primarily interested in learners which exhibit expedient induction. Our reason for this concern will become more apparent when we examine the complexity involved in locating a target concept, even in a small universe. Consider, for example, the difficulty in locating a single concept class in a universe consisting of only four objects. Since the universe of all dichotomous sets (the concept C and U-C) containing n objects can be represented as a lattice structure having 2^n nodes, we see that this is an exponential search problem (Figure 18.1).

18.4 GENERALIZATION AND SPECIALIZATION

In this section we consider some techniques which are essential for the application of inductive learning algorithms. Concept learning requires that a guess or estimate of a larger class, the target concept, be made after having observed only some fraction of the objects belonging to that class. This is essentially a process of generalization, of formulating a description or a rule for a larger class but one which is still

26—

consistent with the observed positive examples. For example, given the three positive instances of objects

(blue cube rigid large)
(small flexible blue cube)
(rigid small cube blue)

a proper generalization which implies the three instances is blue cube. Each of the instances satisfies the general description.

Specialization is the opposite of generalization. To specialize the concept blue cube, a more restrictive class of blue cubes is required such as small blue cube or flexible blue cube or any of the original instances given above. Specialization may be required if the learning algorithm over-generalizes in its search for the target concept. An over-generalized hypothesis is inconsistent since it will include some negative instances in addition to the positive ones.

There are many ways to form generalizations. We shall describe the most commonly used rules below. They will be sufficient to describe all of the learning paradigms which follow. In describing the rules, we distinguish between two basic types of generalization, comparable to the corresponding types of induction (Section 18.2), selective generalization and constructive generalization (Michalski, 1983). Selective generalization rules build descriptions using only the descriptors (attributes and relations) that appear in the instances, whereas constructive generalization rules do not. These concepts are described further below.

Generalization Rules

Since specialization rules are essentially the opposite of rules for generalization, to specialize a description, one could change variables to constants, add a conjunct or remove a disjunct from a description, and so forth. Of course, there are other means of specialization such as taking exceptions in descriptions (a fish is anything that swims in water but does not breathe air as do dolphins). Such methods will be introduced as needed.

These methods are useful tools for constructing knowledge structures. They give us methods with which to formulate and express inductive hypotheses. Unfortunately, they do not give us much guidance on how to select hypotheses efficiently. For this, we need methods which more directly limit the number of hypotheses which must be considered.

Selective Generalization

Changing Constants to Variables.

An example of this rule was given in Section 18.1. Given instances of a description or predicate $P(a_1)$, $P(a_2)$, . . . , $P(a_k)$ the constants a_i are changed to a variable which may be any value in the given domain, that is, $\forall x\ P(x)$.

Dropping Condition.

Dropping one or more conditions in a description has the effect of expanding or increasing the size of the set. For example, the set of all small red spheres is less general than the set of small spheres. Another way of stating this rule when the conjunctive description is given is that a generalization results when one or more of the conjuncts is dropped.

Adding an Alternative.

This is similar to the dropping condition rule. Adding a disjunctive term generalizes the resulting description by adding an alternative to the possible objects. For example, transforming red sphere to (red sphere) V (green pyramid) expands the class of red spheres to the class of red spheres or green pyramids. Note that the *internal disjunction* could also be used to generalize. An internal disjunction is one which appears inside the parentheses such as (red V green sphere).

Climbing a Generalization Tree.

When the classes of objects can be represented as a tree hierarchy as pictured in Figure 18.2, generalization is accomplished by simply climbing the tree to a node which is higher and, therefore, gives a more general description. For example, moving up the tree one level from elephant we obtain the more general class description of mammal. A greater generalization would be the class of all animals.

Closing an Interval.

When the domain of a descriptor is ordered ($d_1 < d_2 < \dots < d_k$) and a few values lie in a small interval, a more restricted form of generalization than changing constants to variables can be performed by generalizing the values to a closed interval. Thus, if two or more instances with values $D = d_i$ and $D = d_j$

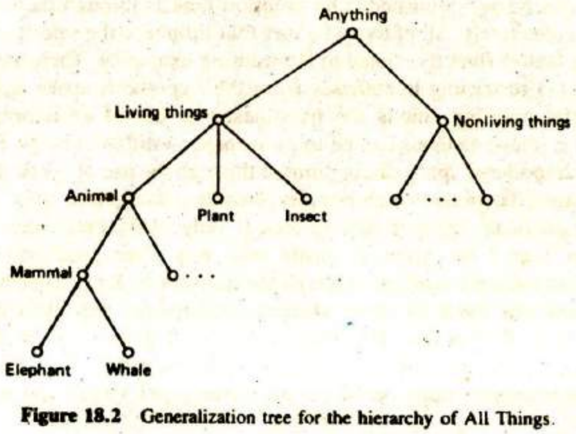


Figure 18.2 Generalization tree for the hierarchy of All Things.

where $d_i < d_j$ have been observed, the generalization $D = [d_i \ldots d_j]$ can be made; that is, D can be any value in the interval d_i to d_j.

Constructive Generalization

Generating Chain Properties.

If an order exists among a set of objects, they may be described by their ordinal position such as first, second, . . . , n^{th}. For example, suppose the relations for a four story building are given as

$$\text{above}(f_2, f_1) \ \& \ \text{above}(f_3, f_2) \ \& \ \text{above}(f_4, f_3)$$

then a constructive generalization is

$$\text{most\_above}(f_4) \ \& \ \text{least\_above}(f_1).$$

The most above, least above relations are created. They did not occur in the original descriptors.

Other forms of less frequently used generalization techniques are also available including combinations of the above. We will introduce such methods as we need them.

18.5 INDUCTIVE BIAS

Learning generalizations has been characterized as a search problem (Mitchell, 1982). We saw in Section 18.2 that learning a target concept is equivalent to finding a node in a lattice of 2^n nodes when there are n elementary objects. How can one expect to realize expedient induction when an exponential space must be searched? From our earlier exposure to search problems, we know that the naive answer to this question is simply to reduce the number of hypotheses which must be considered. But how can this be accomplished? Our solution here is through the use of *bias*.

Bias is, collectively, all of those factors that influence the selection of hypotheses, excluding factors directly related to the training examples. There are two general types of bias: (1) restricting hypotheses from the hypothesis space and (2) the use of a preferential ordering among the hypotheses or use of an informed selection scheme. Each of these methods can be implemented in different ways. For example, the size of the hypothesis space can be limited through the use of syntactic constraints in a representation language which permits attribute descriptions only. This will be the case with predicate calculus descriptions if only unary predicates are allowed, since relations cannot be expressed easily with one place predicates. Of course, such descriptions must be expressive enough to represent the knowledge being learned.

Representations based on more abstract descriptions will often limit the size of the space as well. Consider the visual scene of Figure 18.3. When used as a training example for concepts like on top of, the difference in the size of the hypothesis space between representations based on primitive pixel values and more abstract

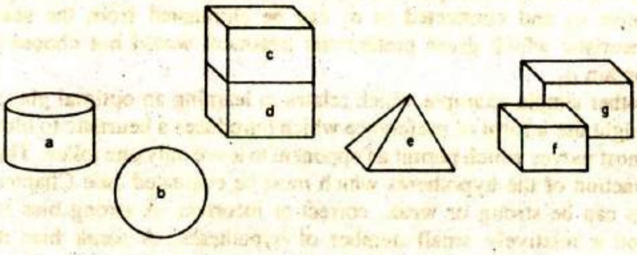


Figure 18.3 Blocks world scene.

descriptions based on a semantic net can be very large. For example, a representation using only two levels of gray (light and dark) in a 1024 by 1024 pixel array will have a hypothesis space in excess of 2^{10^6}. Compare this with the semantic net space which uses no more than 10 to 20 objects and a limited number of position relationships. Such a space would have no more than 10^4 or 10^5 object-position relationships. We see then that the difference in the size of the search space for these two representations can be immense.

Another simple example which limits the number of hypotheses is illustrated in Figure 18.4. The tree representation on the left contains more information and, therefore, will permit a larger number of object descriptions to be created than with the tree on the right. On the other hand, if one is only interested in learning general descriptions of geometrical objects without regard to details of size, the tree on the right will be a superior choice since the smaller tree will result in less search.

Methods based on the second general type of bias limit the search through preferential hypotheses selection. One way this can be achieved is through the use of heuristic evaluation functions. If it is known that a target concept should not contain some object or class of objects, all hypotheses which contain these objects can be eliminated from consideration. Referring again to Figure 18.1, if it is known

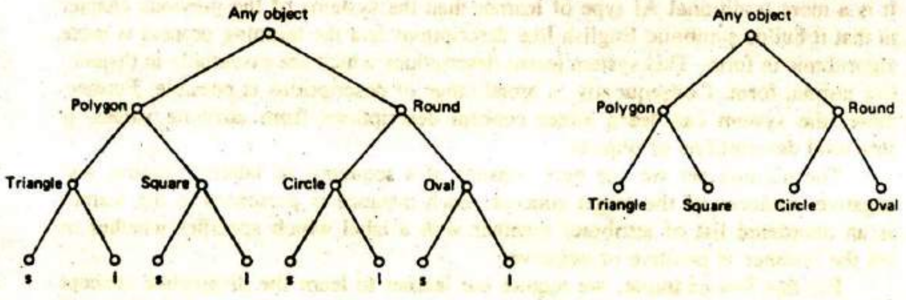


Figure 18.4 Tree representation for object descriptions (s = small, l = large).

that object o_3 (or the description of o_3) should not be included in the target set, all nodes above o_3 and connected to o_3 can be eliminated from the search. In this case, a heuristic which gives preferential treatment would not choose descriptions which contain o_3.

Another simple example which relates to learning an optimal play of the game of Nim might use a form of preference which introduces a heuristic to block consideration of most moves which permit an opponent to leave only one token. This eliminates a large fraction of the hypotheses which must be evaluated (see Chapter 17).

Bias can be strong or weak, correct or incorrect. A strong bias is one which focuses on a relatively small number of hypotheses. A weak bias does not. A correct bias is one which allows the learner to consider the target concept, whereas an incorrect bias does not. Obviously, a learner's task is simplified when the bias is both strong and correct (Utgoff, 1986). Bias can also be implemented in a program as either static or dynamic. When dynamic bias is employed, it is shifted automatically by the program to improve the learner's performance. We will see different forms of bias used in subsequent sections.

18.6 EXAMPLE OF AN INDUCTIVE LEARNER

Many learning programs have been implemented which construct descriptions composed of conjunctive features only. Few have been implemented to learn disjunctive descriptions as well. This is because conjunctive learning algorithms are easier to implement. Of course, a simple implementation for a disjunctive concept learner would be one which simply forms the disjunction of all positive training instances as the target concept. Obviously, this would produce an awkward description if there were many positive instances.

There are many concepts which simply cannot be described well in conjunctive terms only. One of the best examples is the concept of uncle since an uncle can be *either* the brother of the father *or* the brother of the mother of a child. To state it any other way is cumbersome.

The system we describe below was first implemented at M.I.T (Iba, 1979). It is a more traditional AI type of learner than the systems of the previous chapter in that it builds symbolic English-like descriptions and the learning process is more algorithmic in form. This system learns descriptions which are essentially in disjunctive normal form. Consequently, a broad range of descriptions is possible. Furthermore, the system can learn either concept descriptions from attribute values or structural descriptions of objects.

The training set we use here consists of a sequence of labeled positive and negative instances of the target concept. Each instance is presented to the learner as an unordered list of attributes together with a label which specifies whether or not the instance is positive or negative.

For this first example, we require our learner to learn the disjunctive concept "something that is either a tall flower or a yellow object." One such instance of

```
concept_name:(tall flower or yellow object)
  positive_part:
      cluster:description: ____________________
              examples: ____________________
                        ____________________
                                 .
                                 .
                                 .
      cluster:description: ____________________
              examples: ____________________
                        ____________________
                                 .
                  .              .
                  .
                  .
  negative_part:
              examples: ____________________
                        ____________________
```

Figure 18.5 Frame-like knowledge structure.

this concept is represented as (short skinny yellow flower +), whereas a negative instance is (brown fat tall weed −). Given a number of positive and negative training instances such as these, the learner builds frame-like structures with groups of slots we will call clusters as depicted in Figure 18.5.

The target concept is given in the concept name. The actual description is then built up as a group of slots labeled as clusters. All training examples, both positive and negative, are retained in the example slots for use or reuse in building up the descriptions. An example will illustrate the basic algorithm.

Garden World Example

Each cluster in the frame of Figure 18.5 is treated as a disjunctive term, and descriptions within each cluster are treated as conjuncts. A complete learning cycle will clarify the way in which the clusters and frames (concepts) are created. We will use the following training examples from a garden world to teach our learner the concept "tall flower or yellow object."

(tall fat brown flower +)
(green tall skinny flower +)
(skinny short yellow weed +)
(tall fat brown weed −)
(fat yellow flower tall +)

After accepting the first training instance, the learner creates the tentative concept hypothesis "a tall fat brown flower." This is accomplished by creating a cluster in the positive part of the frame as follows:

```
concept_name:(tall flower or yellow object)
 positive_part:
        cluster:description:(tall fat brown flower)
               :examples:(tall fat brown flower)
negative_part: ____________________________
```

With only a single example, the learner has concluded the tentative concept must be the same as the instance. However, after the second training instance, a new hypothesis is created by merging the two initial positive instances. Two instances are merged by taking the set intersection of the two. This results in a more general description, but one which is consistent with both positive examples. It produces the following structure.

```
concept_name:(tall flower or yellow object)
 positive_part:
        cluster:description:(tall, flower)
               :examples:(tall fat brown flower)
                         (green tall skinny flower)
negative_part:
     examples: ____________________________
```

The next training example is also a positive one. Therefore, the set intersection of this example and the current description is formed when the learner is presented with this example. The resultant intersection and new hypothesis is an over generalization, namely, the null set, which stands for anything.

```
concept_name:(tall flower or yellow object)
 positive_part:
        cluster:description:()
               :examples:(tall fat brown flower)
                         (green tall skinny flower)
                         (skinny short yellow weed)
negative_part:
               :examples: ____________________
```

The fourth instance is a negative one. This instance is inconsistent with the current hypothesis which includes anything. Consequently, the learner must revise its hypothesis to exclude this last instance.

It does this by splitting the first cluster into two new clusters which are then both compatible with the negative instance. Each new cluster corresponds to a disjunctive term in this description.

To build the new clusters, the learner uses the three remembered examples from the first cluster. It merges the examples in such a way that each merge produces new consistent clusters. After merging we get the following revised frame.

```
concept_name:(tall flower or yellow object)
 positive_part:
        cluster:description:(tall flower)
                :examples:(tall fat brown flower)
                          (green tall skinny flower)
        cluster:description:(skinny short yellow weed)
                :examples:(skinny short yellow weed)
 negative_part:
                :examples:(tall fat brown weed)
```

The reader should verify that this new description has now excluded the negative instance.

The next training example is all that is required to arrive at the target concept. To complete the description, the learner attempts to combine the new positive instance with each cluster by merging as before, but only if the resultant merge is compatible with all negative instances (one in this case). If the new instance cannot be merged with any existing cluster without creating an inconsistency, a new cluster is created.

Merging the new instance with the first cluster results in the same cluster. Merging it with the second cluster produces a new, more general cluster description of yellow. The final frame obtained is as follows.

```
concept_name:(tall flower or yellow object)
 positive_part:
        cluster:description:(tall flower)
                :examples:(tall fat brown flower)
                          (green tall skinny flower)
                          (fat yellow flower tall)

        cluster:description:(yellow)
                :examples:(skinny short yellow weed)
                          (fat yellow flower tall)
 negative_part:
                :examples:(tall fat brown weed)
```

The completed concept now matches the target concept "tall flower or yellow object."

The above example illustrates the basic cycle but omits some important factors. First, the order in which the training instances are presented to the learner is important. Different orders, in general, will result in different descriptions and may require different numbers of training instances to arrive at the target concept.

Second, when splitting and rebuilding clusters after encountering a negative example, it is possible to build clusters which are not concise or maximal in the sense that some of the clusters could be merged without becoming inconsistent. Therefore, after rebuilding new clusters it is necessary to check for this maximality and merge clusters where possible without violating the inconsistency condition.

Blocks World Example

Another brief example will illustrate this point. Here we want to learn the concept "something that is either yellow or spherical." For this, we use the following training instances from a blocks world.

```
(yellow pyramid soft large +)
(blue sphere soft small +)
(yellow pyramid hard small +)
(green large sphere hard +)
(yellow cube soft large +)
(blue cube soft small −)
(blue pyramid soft large −)
```

After the first three training examples have been given to the learner, the resultant description is the empty set.

```
concept_name:(yellow or spherical object)
 positive_part:
       cluster:description:()
              :examples:(yellow pyramid soft large)
                        (blue sphere soft small)
                        (yellow pyramid hard small)
negative_part:
              :examples: ____________________
```

Since the next two examples are also positive, the only change to the above frame is the addition of the fourth and fifth training instances to the cluster examples. However, the sixth training instance is negative. This forces a split due to the inconsistency. In rebuilding the clusters this time, we rebuild starting with the last positive (fifth) example and work backwards as though the examples were put on a stack. This is actually the order used in the original system. After the fifth and fourth examples are processed, the following frame is produced.

```
concept_name:(yellow or spherical object)
 positive_part:
      cluster:description:(large)
           :examples:(yellow cube soft large)
                    (green large sphere hard)

 negative_part:
           :examples: ____________________
```

Next, when an attempt is made to merge the third example, an inconsistency results. Therefore, this example must be put into a separate cluster. The same applies when an attempt is made to merge the second example with either of the new clusters; merging with the first cluster results in the empty set, while merging with the second cluster results in the set (small) which is also inconsistent with the negative example (blue cube soft small). This forces the creation of a third cluster. Finally, after attempts are made to merge the first example (it merges with the first two clusters), we obtain the frame

```
concept_name:(yellow or spherical object)
 positive_part:
      cluster:description:(large)
           :examples:(yellow cube soft large)
                    :(green large sphere hard)
                    :(yellow pyramid soft large)
      cluster:description:(yellow pyramid)
           :examples:(yellow pyramid hard small)
                    (yellow pyramid soft large)
      cluster:description:(blue sphere soft small)
           :examples:(blue sphere soft small)
negative_part:
           :examples:(blue cube soft small)
```

Note that we still have not arrived at the target concept. The last training instance, a negative one,

(blue pyramid soft large $-$)

is needed to do the trick. The first cluster is inconsistent with this instance. Therefore, it must be split. After completing this split we get the new frame

```
concept_name:(yellow or spherical object)
 positive_part:
       cluster:description:(yellow soft large)
              :examples:(yellow cube soft large)
                        (yellow pyramid soft large)
       cluster:description:(green large sphere hard)
              :examples:(green large sphere hard)
       cluster:description:(yellow pyramid)
              :examples:(yellow pyramid hard small)
                        (yellow pyramid soft large)
       cluster:description:(blue sphere soft small)
              :examples:(blue sphere soft small)
negative_part:
              :examples:(blue cube soft small)
                        (blue pyramid soft large)
```

All of the new clusters are now consistent with the negative examples. But the clusters are not maximal, since it is possible to merge some clusters without violating the inconsistency condition. To obtain maximality, the clusters must be rewritten and merged where possible. This rewrite is accomplished by copying the first cluster and then successively merging or copying the other clusters in combination. Of course, a merge can be completed only when an inconsistency does not result.

The first two clusters cannot be merged as we know from the above. The first and third clusters can be merged to give yellow. The second and fourth clusters can also be merged to produce sphere. These are the only merges that can be made that are compatible with both negative examples. The final frame then is given as

```
concept_name:(yellow or spherical object)
 positive_part:
       cluster:description:(yellow)
              :examples:(yellow cube soft large)
                        (yellow pyramid hard
                        small)
                        (yellow pyramid soft
                        large)
       cluster:description:(sphere)
              :examples:(green large sphere hard)
                        (blue sphere soft small)
 negative_part:
              :examples:(blue cube soft small)
                        (blue pyramid soft large)
```

It may have been noticed already by the astute reader that there is no reason why negative-part clusters could not be created as well. Allowing this more symmetric structure permits the creation of a broader range of concepts such as ''neither yellow nor spherical'' as well as the positive type of concepts created above. This is implemented by building clusters in the negative part of the frame using the negative examples in the same way as the positive examples. In building both descriptions concurrently, care must be taken to maintain consistency between the positive and negative parts. Each time a negative example is presented, it is added to the negative part of the model, and a check is made against each cluster in the positive part of the model for inconsistencies. Any of the clusters which are inconsistent are split into clusters which are maximal and consistent and which contain all the original examples among them. We leave the details as an exercise.

Network Representations

It is also possible to build clusters of network representation structures and to learn structural descriptions of objects. For example, the concept of an arch can be learned in a manner similar to the above examples. In this case our training examples could be represented as something like the following where, as in earlier chapters, ako means a kind of.

```
((nodes (a b c)
 (links (ako a brick)
        (ako b brick)
        (ako c brick)
        (supports a c)
        (supports b c) +)
```

Since an arch can support materials other than a brick, another positive example of the concept arch might be identical to the one above except for the object supported, say a wedge. Thus, substituting (ako c wedge) for (ako c brick) above we get a second positive instance of arch. These two examples can now be generalized into a single cluster by simply dropping the differing conjunctive ako terms to get the following.

```
((nodes (a b c)
 links  (ako a brick)
        (ako b brick)
        (supports a c)
        (supports b c))
```

This is an over generalization. It can be corrected by a negative training example which uses some nonvalid object such as a sphere as the supported item.

```
((nodes (a b c)
 links  (ako a brick)
        (ako b brick)
        (ako c sphere)
        (supports a c)
        (supports b c) -)
```

This example satisfies the current description of an arch. However, it has caused an inconsistency. Therefore, the cluster must be split into a disjunctive description as was done before in the previous examples. The process is essentially the same except for the representation scheme.

In a similar manner the concept of uncle can be learned with instances presented and corresponding clusters created using a network representation as follows.

```
((nodes (a b c)
 links  (ako a person)
        (ako b person)
        (ako c person)
        (male c)
        (parent_of b a)
        (brother_of c b) +)
```

Again, we leave the remaining details as an exercise.

Before leaving these examples, the reader should consider what learning methods and tools have been used from the previous two sections, what types of bias have been used to limit the search, and what methods of generalization have been employed in the learning process.

18.7 SUMMARY

Inductive learning is accomplished through inductive inference, the process of inferring a common class from instances, reasoning from parts to the whole or from the individual to the universal. Learning a concept through induction requires generalization, a search through a lattice of hypotheses. Practical induction is based on the use of methods that constrain or direct the search process. This is made possible through the use of bias which is used to reduce the size of the hypothesis space or to impose preferential selection on the hypotheses.

A number of techniques are available for either selective or constructive generalization, including changing constants to variables, dropping conjunctive conditions, adding a disjunctive alternative, closing the interval, climbing a generalization tree, and generating chain properties, among others. Specialization is achieved through the inverse operation to generalization as well as some other methods like including exceptions.

An example of an inductive learning system was presented in some detail. This system constructs concept descriptions from positive and negative examples presented as attribute lists. The descriptions are created as clusters or disjuncts by first generalizing conjunctive descriptions from the positive training examples until an inconsistent negative example is experienced. This separates the clusters to produce compatible, disjunctive descriptions. The system can also learn structural descriptions in the form of network representations. It is possible to build negative disjunctive descriptions as well, building clusters in the negative part of the frame structure or both positive and negative descriptions.

EXERCISES

18.1. Define inductive learning and explain why we still use it even though it is not a "valid" form of learning.

18.2. What is the difference between a class and a concept?

18.3. What is the difference between selective, constructive, and expedient induction? Give examples of each.

18.4. What is the purpose of inductive bias?

18.5. Give three examples in which inductive bias can be applied to constrain search.

18.6. Use the following training examples to simulate learning the concept "green flower or skinny object." Build up the concept description in clusters using the same method as that described in Section 18.6.

(green tall fat flower +)
(skinny green short flower +)
(tall skinny green flower +)
(red skinny short weed +)
(green short fat weed −)
(tall green flower skinny +)

18.7. Work out an example of concept learning using network structures. The concept to be learned is the concept wife. Create both positive and negative training examples.

18.8. Write a computer program in LISP to build concept descriptions in the form of clusters like the examples of Section 18.7.

18.9. The method described in Section 18.7 for learning concepts depends on the order in which examples are presented. State what modifications would be required to make the learner build the same structures independent of the order in which training examples are presented.

18.10. Compare each of the generalization methods described in Section 18.4 and explain when each method would be appropriate to use.

18.11. Referring to the previous problem, rank the generalization methods by estimated computation time required to perform each.

18.12. Give an example of learning the negative of the concept "tall flower or red object," that is, something that is "neither a tall flower nor a red object."

19

Examples of Other Inductive Learners

19.1 INTRODUCTION

In this chapter we contnue with our study of inductive learning. Here, we review four other important learning systems based on the inductive paradigms.

Perhaps the most significant difference among these systems is the type of knowledge representation scheme and the learning algorithms used. The first system we describe, ID3, constructs a discrimination tree for use in classifying objects. The second system, LEX, creates and refines heuristic rules for carrying out symbolic integrations. The third system, INDUCE, constructs descriptions in an extended form of predicate calculus. These descriptions are then used to classify objects such as soybean diseases. Our final system, Winston's Arch, forms conjunctive network structures similar to the ones described in the previous chapter.

19.2 THE ID3 SYSTEM

ID3 was developed in the late 1970s (Quinlan, 1983) to learn object classifications from labeled training examples. The basic algorithm is based on earlier research programs known as Concept Learner Systems or CLSs (Hunt et al., 1966). This system is also similar in many respects to the expert system architecture described in Section 15.3.

The CLS algorithms start with a set of training objects $O = \{o_1, o_2, \ldots, o_n\}$ from a universe U, where each object is described by a set of m attribute values. An attribute A_j having a small number of discrete values $a_{j1}, a_{j2}, \ldots, a_{jk}$ is selected and a tree node structure is formed to represent A_j. The node has k_j branches emanating from it where each branch corresponds to one of the a_{ji} values (Figure 19.1). The set of training objects O are then partitioned into at most k_j subsets based on the object's attribute values. The same procedure is then repeated recursively for each of these subsets using the other $m - 1$ attributes to form lower level nodes and branches. The process stops when all of the training objects have been subdivided into single class entities which become labeled leaf nodes of the tree.

The resulting discrimination tree or decision tree can then be used to classify new unknown objects given a description consisting of its m attribute values. The unknown is classified by moving down the learned tree branch by branch in concert with the values of the object's attributes until a leaf node is reached. The leaf node is labeled with the unknown's name or other identity.

ID3 is an implementation of the basic CLS algorithm with some modifications. In the ID3 system, a relatively small number of training examples are randomly selected from a large set of objects O through a window. Using these training examples, a preliminary discrimination tree is constructed. The tree is then tested by scanning all the objects in O to see if there are any exceptions to the tree. A new subset or window is formed using the original examples together with some of the exceptions found during the scan. This process is repeated until no exceptions are found. The resulting discrimination tree can then be used to classify new objects.

Another important difference introduced in ID3 is the way in which the attributes are ordered for use in the classification process. Attributes which discriminate best are selected for evaluation first. This requires computing an estimate of the expected information gain using all available attributes and then selecting the attribute having the largest expected gain. This attribute is assigned to the root node. The attribute having the next largest gain is assigned to the next level of nodes in the tree and so on until the leaves of the tree have been reached. An example will help to illustrate this process.

For simplicity, we assume here a single-class classification problem, one where all objects either belong to class C or U-C. Let h denote the fraction of objects

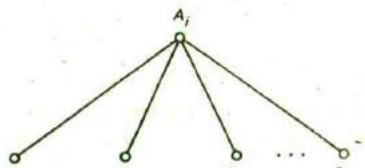


Figure 19.1 A node created for attribute A_j (color) with k discrete values, $a_{j1}, a_{j2}, \ldots, a_{jk}$ (red, orange, . . . , white).

that belong to class C in a sample of n objects from O; h is an estimate of the true fraction or probability of objects in U that belong to class C. Also let

c_{jk} = number of objects that belong to C and have value a_{jk}

d_{jk} = number of objects *not* belonging to C and having value a_{jk}

$p_{jk} = (c_{jk} + d_{jk}) / n$ be the fraction of objects with value a_{jk} (we assume objects have all attribute values so that $\Sigma_j\, p_{jk} = 1$)

$f_{jk} = c_{jk} / (c_{jk} + d_{jk})$, the fraction of objects in C with attribute value a_{jk}, and $g_{jk} = 1 - f_{jk}$, the fraction of objects *not* in C with value a_{jk}.

Now define

$$H_c(h) = -h * \log_2 h - (1 - h) * \log_2(1 - h)$$

with $H_c(0) = 0$, and

$$H_{jk} = -f_{jk} * \log_2 f_{jk} - g_{jk} * \log_2 g_{jk}$$

as the information content for class C and attribute a_{jk} respectively. Then the expected value (mean) of H_{jk} is just

$$E(H_{jk}) = \Sigma p_{jk} * H_{jk}$$

We can now define the gain G_j for attribute A_j as

$$G_j = H_c - H_{jk}$$

Each G_j is computed and ranked. The ones having the largest values determine the order in which the corresponding attributes are selected in building the discrimination tree.

In the above equation, quantities computed as

$$H = -\Sigma p_i * \log_2 p_i \quad \text{with} \quad \Sigma_i p_i = 1$$

are known as the information theoretic entropy where p_i is the probability of occurrence of some event i. The quantities H provide a measure of the dispersion or surprise in the occurrence of a number of different events. The gains G_j measure the information to be gained in using a given attribute.

In using attributes which contain much information, one should expect that the size of the decision tree will be minimized in some sense, for example, in total number of nodes. Therefore, choosing those attributes which contain the largest gains will, in general, result in a smaller attribute set. This amounts to choosing those attributes which are more relevant in characterizing given classes of objects.

In concluding this section, an example of a small decision tree for objects described by four attributes is given in Figure 19.2. The attributes and their values are horned = {yes, no}, color = {black, brown, white, grey}, weight = {heavy, medium, light}, and height = {tall, short}. One training set for this example consists

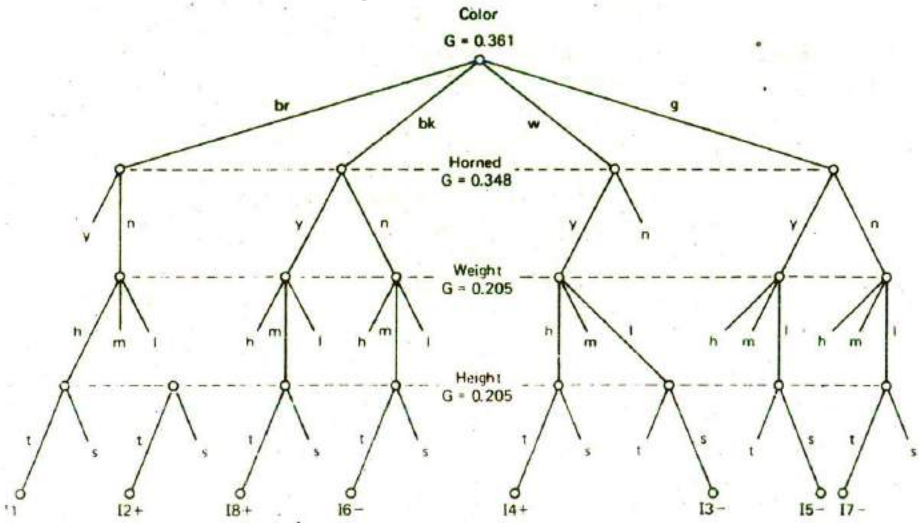


Figure 19.2 Discrimination tree for three attributes ordered by information gain. Abbreviations are br = brown, bk = black, w = white, g = gray, y = yes, n = no, h = heavy, m = medium, l = light, t = tall, and s = short.

of the eight instances given below where members of the class C have been labeled with + and nonmembers with − (Class C might, for example, be the class of cows).

I1 (brown heavy tall no −)
I2 (black heavy tall yes +)
I3 (white light short yes −)
I4 (white heavy tall yes +)
I5 (grey light short yes −)
I6 (black medium tall no −)
I7 (grey heavy tall no −)
I8 (black medium tall yes +)

The computations required to determine the gains are tabulated in Table 19.1. For example, to compute the gain for the attribute color, we first compute

$$H_{t} = -3/8 * \log_2 3/8 - 5/8 * \log_2 5/8 = 0.955$$

TABLE 19.1 SUMMARY OF COMPUTATIONS REQUIRED FOR THE GAIN VALUES.

		c_{jk}	$c_{jk} + d_{jk}$	f_{jk}	$\log f_{jk}$	$\log g_{jk}$	I_{jk}	G_j
$k = 1$	brown	0	1	0	—	—	0	
2	black	2	3	2/3	−0.585	−1.585	0.918	
3	white	1	2	1/2	−1.0	−1.0	1.0	0.361
4	grey	0	2	0	—	—	0	
$k = 1$	tall	3	6	1/2	−1.0	−1.0	1.0	
2	short	0	2	0	—	—	0	0.205
$k = 1$	heavy	2	4	1/2	−1.0	−1.0	1.0	
2	medium	1	2	1/2	−1.0	−1.0	1.0	0.205
3	light	0	2	0	—	—	0	
$k = 1$	yes	3	5	3/5	−0.737	−1.322	0.971	
2	no	0	3	0	—	—	0	0.348

The information content for each color value is then computed.

$$H(\text{brown}) = 0$$

$$H(\text{black}) = -2 / 3 * \log_2 2 / 3 - 1 / 3 * \log_2 1 / 3 = 0.918$$

$$H(\text{white}) = -1 / 2 * \log_2 1/ 2 - 1 / 2 * \log_2 1 / 2 = 1.0$$

$$H(\text{grey}) = 0$$

$$E(H_{\text{color}}) = 1 / 8 * 0 + 3 / 8 * 0.918 + 1 / 4 * 1.0 + 0 = 0.594.$$

Therefore, the gain for color is

$$G = H_c - H_{\text{color}} = 0.955 - 0.594 = 0.361$$

The other gain values for horned, weight, and height are computed in a similar manner.

19.3 THE LEX SYSTEM

LEX was developed during the early 1980s (Mitchell et al., 1983) to learn heuristic rules for the solution of symbolic integration problems. The system is given about 40 integration operators which are expressed in the form of rewrite rules. Some of the rules are shown in Figure 19.3a. Internal representations for some typical integral expressions are given in Figure 19.3b.

Each of the operators has preconditions which must be satisfied before it can be applied. For example, before OP6 can be applied, the general form of the integrand must be the product of two real functions, that is $udv = f_1(x) * f_2(x)dx$. Each operator also has associated with it the resultant states that can be produced by that operator. For example, OP6 can have

OP1	$1 * f(x) \rightarrow f(x)$ ($f(x)$ is any real function of x)
OP2	$\int r * f(x) \rightarrow r \int f(x)\,dx$ (r is any real number)
OP3	$\int \sin(x)\,dx \rightarrow -\cos(x)\,dx$
OP4	$\int \cos(x)\,dx \rightarrow \sin(x)$
OP5	$\int [f_1(x) + f_2(x)]dx \rightarrow \int f_1(x)\,dx + \int f_2(x)\,dx$
OP6	$\int u\,dv \rightarrow uv - \int v\,du$ (integration by parts)
⋮	⋮

Figure 19.3a Typical calculus operators.

$\int[\sin(x) + \cos(x)]dx$	(int((+ sin cos) x))
$\int \cos^2(x)\,dx$	(int(↑ cos 2) x))
$\int x * e^x\,dx$	(int((* id(↑ e id)) x))

Figure 19.3b Typical calculus representations.

the result obtained from the opposite bindings. The choice of the result obtained with u bound to f_1 and dv to f_2 or of an incorrect or poor operator at a given stage in the solution will lead to failure or possibly to a lengthy solution. The learning problem then is to create or refine heuristic rules which suggest when an operator should be used.

All heuristics in LEX are of the form

If: The integrand pattern is P •
Then: Apply OPn with bindings B.

For example, a typical heuristic for OP6 would be

$\int f(x) * \text{trig}(x)\,dx \rightarrow$ Apply OP6 with bindings $u = f(x)$, and $dv = \text{trig}(x)\,dx$

Part of the refinement problem is the generalization or specialization of the heuristics to apply to as many consistent instances as possible. Generalization and specialization are achieved in LEX through the use of a hierarchical description tree. A segment of this tree is depicted in Figure 19.4. Thus, when a rule applies to more than a single trig function such as to both sin and cos, the more general term trig would be substituted in the rule. Likewise, when a rule is found which applies to both log and exp functions, the exp_log description would be used.

LEX is comprised of four major components as illustrated in Figure 19.5. The Problem Generator selects and generates informative integration problems which are submitted to the Problem Solver for solution. The Generator was included as part of the system to provide well-ordered training examples and to make the system a fully automatic learner. The Problem Solver attempts to find a solution to this problem using available heuristics and operators. (A solution has been found when an operator produces an expression not containing an integral.)

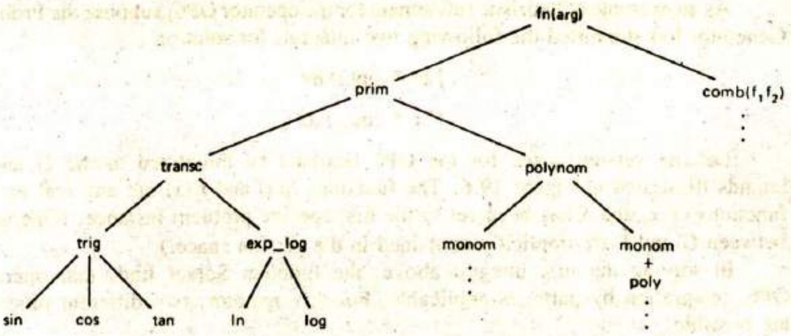


Figure 19.4 A segment of the LEX generalization tree grammar.

Output from the Problem Solver is some solution together with a complete trace of the solution search. This is presented to the Critic unit for evaluation. The Critic then analyzes the solution trace, comparing it to a least-cost path and passes related positive or negative training instances to the Generalizer. A positive instance is an operator which lies on the least-cost path, while a negative instance is one lying off the path. Given these examples, the Generalizer modifies heuristics to improve the selection of operators for best application during an attempted solution.

During the learning process each operator is given a *version space* of heuristic rules associated with it. Rules in the version space give the conditions under which the operator applies as well as the domain of states that can be produced. The version space is actually stored as two bounding rules, a rule G which gives the most general conditions of application and a rule S which gives the most specific conditions. Between these two bounds are implicitly contained all plausible versions of the heuristic. As the system learns, the bound S is made more general to include all positive instances presented by the Critic while the bound G is made more specific to exclude all negative instances. When the two bounds become equal ($G = S$), the correct heuristic has been learned.

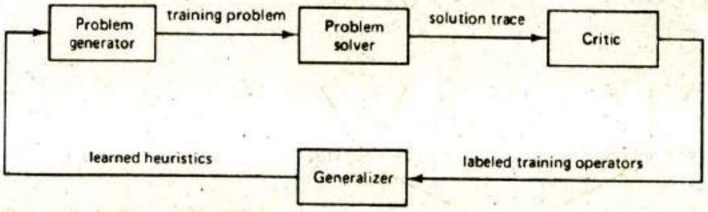


Figure 19.5 LEX learning model.

As an example of heuristic refinement for the operator OP6, suppose the Problem Generator has submitted the following two integrals for solution.

$$\int 2x * \sin(x)\, dx$$

$$\int 2x * \cos(x)\, dx$$

Let the version space for the OP6 heuristic be initialized to the G and S bounds illustrated in Figure 19.6. The functions $f_1(x)$ and $f_2(x)$ are any real-valued functions of x, and S has been set to the first specific problem instance. (Operators between G and S are implicitly contained in the version space.)

In solving the first integral above, the Problem Solver finds that operator OP6, integration by parts, is applicable. For this operator, two different bindings are possible

$$(a)\quad u = 2x \qquad\qquad (b)\quad u = \sin(x)$$
$$dv = \sin(x)\, dx \qquad\qquad dv = 2x\, dx$$

If the bindings given by a are used, OP6 produces the new expression

$$2x * (-\cos(x)) - \int 2 * (-\cos(x))dx$$

which can be further reduced using OP2, OP4 and other simplification operators to give the correct solution

$$-2x * \cos(x) + 2 * \sin(x) + C$$

For this binding, the Critic will label this as a positive instance.

On the other hand, if the variable bindings given in b are used, OP6 produces the more complex expression

$$x^2 * \sin(x) - \int x^2 * \cos(x)\, dx$$

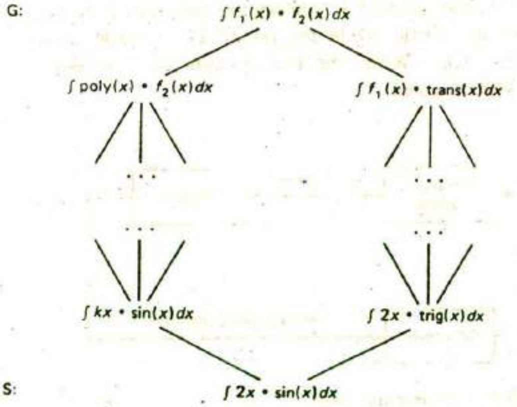


Figure 19.6 Version space with bounds G and S.

In this case the Critic will label that instance as a negative one. This negative instance will be used to adjust the version space to exclude this instance by specializing the G integrand to either

$$\text{poly}(x) * f_2(x)\,dx \qquad \text{or to} \qquad f_1(x) * \text{tran}(x)\,dx$$

with corresponding bindings

$$\begin{aligned} u &= \text{poly}(x) \\ dv &= f_2(x)\,dx \end{aligned} \qquad \text{and} \qquad \begin{aligned} u &= f_1(x) \\ dv &= \text{tran}(x)\,dx \end{aligned}$$

both of which exclude the negative instance.

After the Problem Solver attempts to solve the second integral, $\int 2x * \cos(x)\,dx$, the Critic labels the binding $u = 2x$, $dv = \cos(x)\,dx$ as positive. However, this example is not yet included in the version space. Consequently, S must be generalized to include this instance. The bound S is generalized by finding the least general function in the description hierarchy which includes both sin and cos (Figure 19.4). This generalization produces the new version space

$$G\colon \int f_1(x) * \text{tran}(x)\,dx$$

$$S\colon \int 2x * \text{trig}(x)\,dx$$

Through repeated attempts at solving such well organized problems, LEX is able to create and refine heuristics for each operator which designate when that operator should be applied. Thus, the refined heuristics reduce the class of problems to those producing acceptable solutions.

19.4 THE INDUCE SYSTEM

Several versions of the INDUCE system were developed beginning in the late 1970s (Larson and Michalski, 1977, and Dietterich and Michalski, 1981). INDUCE is a system which discovers similar patterns among positive examples of objects and formulates generalized descriptions which characterize the class patterns. The description language and the internal representation used in the system are an extension of first order predicate calculus. This is one of the unique features of INDUCE.

Before outlining how the system operates, we introduce a few new terms:

Selector. A relational statement defined by either a predicate or a function together with the values it assumes. Typical selectors used to describe objects or events in INDUCE are [shape = square V rectangular], [color = green], [size <= 4], [number_spots = 3 . . 8], [ontop(a,b)], and so on.

Complex. A logical product of selectors. A complex is a conjunctive description of some object (each selector in brackets is regarded as a conjunct). For example, a plant leaf might be represented as

```
Leaf_1 [contains(Leaf_1,spots)][color(Leaf_1) = yellow]&
[shape(Leaf_1 = curled)][length(Leaf_1) = 4][width(Leaf_1 = 2.3]
```

This expression states that a particular leaf contains spots, is yellow, curled at the sides, is 4 cm in length and 2.3 cm wide.

Cover. A cover of a set of events E is a set of complexes which describe or contain all events in E. A cover of the set E_1 against the set E_2, where E_1 and E_2 are disjointed is a set of complexes which cover all events in E_1 and no event in E_2. For example, the complex [shape = circle] [color = red] [size < 5] covers the set of all red circles of size less than 5 and none of the set of circles greater than 5.

Characteristic description. A description of a class of objects which uses descriptive facts that are true for all objects in the class and thus discriminates between objects in the class and all other objects. A characteristic description of a house is one which sets houses apart from all nonhouses.

Discriminant description. A description of a class given in the context of a fixed set of other classes. A fruit tree that bears apples is a discriminant description of an apple tree within the context of fruit trees.

Given specifications for object descriptions and the type of inductive assertions desired, INDUCE can discover and formulate general class descriptions of the objects. We give here only a brief outline of the basic steps followed when inducing a single-class description.

1. The user initially makes a determination of all descriptor types (attributes or relations) needed to describe the objects. This requires a specification of type, the domains, and if appropriate, the conditions of applicability. For example, in describing living cells, number_of_tails would apply only to cells with objects possessing tails. Next, each object is described using the given descriptors and the class to which it belongs. In addition, general rules and descriptors which apply for constructive induction are specified, and finally, the type of output description desired, the characteristic and/or discriminant descriptors.

2. Given the above information, INDUCE breaks down the object descriptions into new descriptors or complexes by dropping a single selector from each description; it then places them in a list. Each of these new structures represents a generalization of the object description (dropping condition rule). Clearly, some of these new descriptors will cover negative objects as well. Consequently, the descriptors are ordered giving highest rank to ones that cover the greatest number of positive objects and the fewest number of negative objects.

3. New descriptions are also created by applying inference rules to the original ones. The inference rules use constructive generalization, heuristics, and other infor-

mation to create the new descriptors. These are then added to the ranked list at their appropriate locations.

4. Each of the descriptions in the list is then tested for consistency and completeness. A description is consistent if it does not cover any negative object. It is complete if it covers all the (positive) objects. Those that pass the test are removed from the ranked list and put on a solutions list. Incomplete but consistent descriptions are put on a consistent list. Any descriptors remaining are specialized by appending selectors from the original list. These modified descriptions are tested for consistency and completeness and the process is repeated until predefined list size limits are exceeded or until all descriptors have been put on either the solutions or the consistent list.

5. Each of the descriptors on the consistent list is made more generic using generalizations such as climbing a generalization tree or closing an interval. The generalizations are then ranked and pruned using a Lexicographic Evaluation Function (LEF) and the best *m* of these are chosen as the description. The LEF uses criteria established by the user such as maximum examples covered, simplest decriptions (fewest terms), or user defined least cost. The final descriptions on the solutions list are the induced (generalized) descriptions which cover all the training instances. The following example will illustrate this process.

Assume the following three descriptions have been given for the object instances displayed in Figure 19.7.

$$\exists o_1, o_2\ [\text{color}(o_1) = \text{green}][\text{shape}(o_1) = \text{sphere}][\text{size}(o_1) = \text{large}]$$
$$[\text{color}(o_2) = \text{red}][\text{shape}(o_2) = \text{box}][\text{size}(o_2) = \text{large}]$$
$$[\text{supports}(o_2, o_1)]$$

$$\exists o_3, o_4\ [\text{color}(o_3) = \text{red}][\text{shape}(o_3) = \text{cylinder}][\text{size}(o_3) = \text{small}]$$
$$[\text{color}(o_4) = \text{red}][\text{shape}(o_4) = \text{cube}][\text{size}(o_4) = \text{large}]$$
$$[\text{supports}(o_4, o_3)]$$

$$\exists o_5, o_6\ [\text{color}(o_5) = \text{blue}][\text{shape}(o_5) = \text{pyramid}][\text{size}(o_5) = \text{small}]$$
$$[\text{color}(o_6) = \text{red}][\text{shape}(o_6) = \text{cylinder}][\text{size}(o_6) = \text{large}]$$
$$[\text{supports}(o_6, o_5)]$$

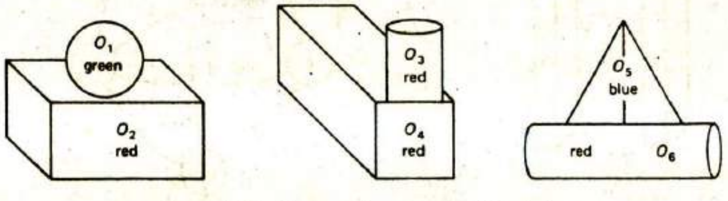


Figure 19.7 Three positive examples for training.

Using the procedure outlined above, INDUCE would discover the generalized description for the examples as "a red object supports another object," that is

$$\exists x,y\ [\text{color}(x) = \text{red}][\text{supports}(x\ y)]$$

19.5 LEARNING STRUCTURAL CONCEPTS

Winston's Arch system was developed early in 1970 (Winston, 1975). This work has been noted as one of the most influential projects in recent AI research and has been cited as being responsible for stimulating renewed research in the area of machine learning.

The Arch system learns concepts in the form of associative network representations (Figure 19.8) much like the cluster network representations of the previous chapter. The Arch system, however, is not able to handle disjunctive descriptions.

Given positive and negative training examples like the ones in Figures 19.9 and 19.10, the system builds a generalized network description of an arch such as

```
(if (and (has_parts x (x1 x2 x3))
        (ako brick x1)
        (ako brick x2)
        (ako prism x3)
        (supports x1 x3)
        (supports x2 x3))
  (isa x arch))
```

Each training example is presented as a blocks world line drawing which is converted to a network representation. The first positive example is taken as the

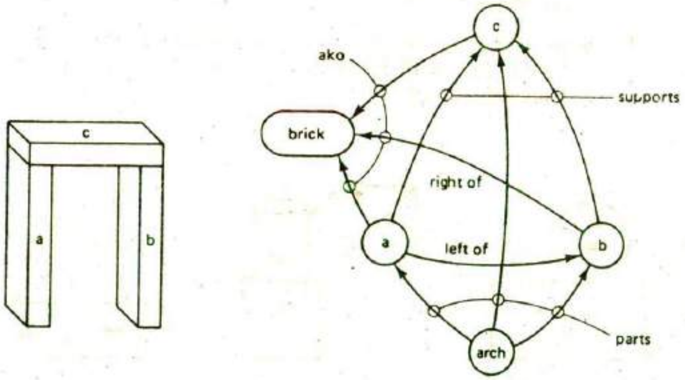
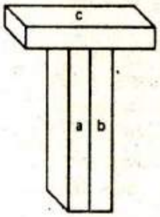


Figure 19.8 An arch and its network representation.

```
((arch(nodes(a b c)         (arch(nodes(a b c)          (arch(nodes a b c)
  (links(ako a brick)         (links(ako a brick)          (links(ako a brick)
     (ako b brick)                (ako b brick)                (ako b brick)
     (ako c brick)                (ako c wedge)                (ako c prism)
     (supports a c)         +     (supports a c)      ⇒        (supports a c)
     (supports b c)               (supports b c)               (supports b c)
     (right_of b a)               (right_of b a)               (right_of b a)
     (left_of a b)))              (left_of a b)))              (left_of a b)))

  Positive Example             Positive Example             Generalization
```

Figure 19.9 Generalized description for two positive examples.



Negative example – near miss

```
==>   ((arch (nodes (a b c)
              (links  (ako a brick)
                      (ako b brick)
                      (ako c prism)
                      (supports a c)
                      (supports b c)
                      (right_of b a)
                      (left_of a b)
              (must_not_touch a b)))

      Specialized exception
```

Figure 19.10 A nonarch and the resulting specialized representation.

initial concept description. The next example is then matched against this description using a graph-matching algorithm. This produces a common subgraph and a list of nodes and links which differ. The unmatched nodes and links are tagged with comments which are used to determine how the current description should be modified. If the new example is positive, the description is generalized (Figure 19.9) by either dropping nodes or links or replacing them with more generalized ones obtained from a hierarchical generalization tree. If the new example is a negative one, the description is specialized to exclude that example (Figure 19.10).

The negative examples are called *near misses*, since they differ from a positive example in only a single detail. Note the form of specialization used in Figure 19.10. This is an example of specialization by taking exception. The network representation for these exceptions are *must* and *must not* links to emphasize the fact that an arch must not have these features.

19.6 SUMMARY

Examples of four different inductive learning paradigms were presented in this chapter. In the first paradigm, the ID3 system, classifications were learned from a set of positive examples only. The examples were described as attribute values of objects. The classifications were learned in the form of discrimination tree. Once created, the ID3 system used the tree to classify new unknown objects. Attributes are selected on the basis of the information Gain expected. This results in a minimal tree size. LEX, the second system described, learned heuristics to choose when certain operators should be used in symbolic integration problems. One of the interesting features of LEX is the use of a version space which bounds the set of plausible heuristics that are applicable in a given problem state. LEX uses a syntactic form of bias, its grammar, to limit the size of the hypothesis space. In LEX, generalizations are found by climbing a hierarchical description tree.

The third system considered, INDUCE, formed generalized descriptions of a class of objects in an extended form of predicate calculus. This system builds both attribute and structural types of descriptions. One weakness of this system, however, is the amount of processing required when creating descriptions by removing the selectors in all possible ways. When the number of object descriptions becomes large, the computation becomes excessive.

The fourth and final inductive learner described in this chapter was Winston's Arch. This system builds associative net representations of structural concepts. One of the unique aspects of this system is the use of near miss negative examples which differ from positive examples in only a single feature. This simplifies the learning process somewhat. This system is similar in some ways to the one described in the previous chapter. It is not able to build disjunctive descriptions like that system, however.

Similar features shared by all of these systems include the use of symbolic representations, and the methods of generalization and specialization. The principal differences among these paradigms are the forms of symbolic representation schemes used and the algorithms employed for generalizing and specializing.

EXERCISES

19.1. Derive the discrimination tree of Figure 19.2 using attributes arranged in the following order: height, weight, horned, and color.

19.2. Prove that the entropy H is maximum when $p_i = p_j$ for all $i, j = 1, 2, \ldots, n$, where

$$H = -\Sigma_i\, p_i * \log_2 p_i$$

19.3. Plot the entropy as a function of p for the case $n = 2$; that is, plot H as p ranges from 0 to 1.

19.4. Describe how LEX generalizes from tan, cos, ln, and log to transc (transcendental).

19.5. Use the concepts related to version space bounding to illustrate how a system can learn the concept of a large circle from both positive and negative examples of objects described by the attributes shape (circle, square, triangle), size (small, medium, large), and color (red, blue, green).

19.6. What is the difference between a standard disjunction and an internal disjunction?

19.7. What is the significance of a cover, and how does it relate to a concept?

19.8. What steps, if any, are taken by INDUCE to avoid overgeneralization?

19.9. Compare the methods of induction used by ID3, LEX and INDUCE. Which of the methods is most efficient? Which is most robust? Which has greatest scope; that is, which is most domain independent?

19.10. Compare Winston's Arch learning system with the system developed by Iba (see Chapter 18). Which is most versatile?

19.11. What is the inductive leap used in inductive learning? Why is it potentially dangerous, but still useful? At what point can it be taken?

20
Analogical and Explanation-Based Learning

In this last chapter on autonomous acquisition, we investigate two of the most promising approaches to artificial learning, analogical and explanation-based learning. These approaches are both potentially powerful forms of learning, yet to date, they have received less attention than other methods. We expect this will change because of the great promise they both offer.

20.1 INTRODUCTION

Unlike inductive or similarity-based learning which is based on the observance of a number of training examples, analogical and explanation-based learning (EBL) can be accomplished with only a single example. In analogical learning, one example in the form of a known solution or a past experience is often sufficient knowledge for learning a new solution. In EBL, one positive training example is all that is needed to develop an explanation of a target concept. Of course, multiple examples may be used in both of these learning types, but in general, only a single example is required.

Analogies play an important role in our reasoning processes. We frequently explain or justify one phenomenon with another. A previous experience can often serve as a framework or pattern for a new, analogous experience. We use the familiar experience as a guide in dealing with the new experience. And, since so many of our acts are near repetitions of previous acts, analogical learning has gained a prominant place in our learning processes.

EBL methods of learning differ from other methods in that the learned knowledge is valid knowledge. It is derived from a set of facts through a deductive reasoning process and, therefore, is justified, consistent knowledge. These EBL methods will most likely find use in conjunction with other learning paradigms where it is important to validate newly learned knowledge.

20.2 ANALOGICAL REASONING AND LEARNING

Analogies are similarities or likenesses between things otherwise different. Things which are similar in some respects tend to be similar in other respects. The things or participants in analogies are unlimited. They may be physical objects, concepts, problems and their solutions, plans, situations, episodes, and so forth.

Analogies play a dominant role in human reasoning and learning processes. Previously remembered experiences are transformed and extended to fit new unfamiliar situations. The old experiences provide scenarios or explanations which tend to match the new situation in some aspects and, therefore, offer the promise of suggesting a solution to the new situation. The old and new situations need not be in the same domain. Frequently, the two domains, the base and target domains, will be entirely different, but similarities between the relationships of objects remain strong, a likely consequence of the regularity of nature. For example, consider the solar planetary system and the Rutherford model of an atom or the flow of liquids in pipes and the flow of electricity in conducting wires. In the first case, both consist of a system of smaller bodies being attracted to and revolving around a more massive nucleus. In the second case, the flow behaviors are governed by the force of the sources, the resistance of the medium, and other properties of the respective flow in networks.

Analogical reasoning is probably the form of reasoning we are most dependent upon for all of our decisions and actions. Its use spans such mundane tasks as finding one's way home from work, to more complex tasks such as playing chess or writing a technical paper. As suggested by Carbonell and Minton (1983), this form of reasoning requires less cognitive effort than more formal types of reasoning, which could explain why analogical reasoning is so prevelant in human thought processes.

Analogies appear in different guises and at varied levels of abstraction. Simpler analogies are the word-object or geometric ones often found in SAT or GRE tests. They take the form

$A \sim B$ (A is like B) or more generally
$A:B :: C:D$ (A is to B as C is to D)

where one of the components is missing. For example, the type of word-object and geometrical analogies typically found in aptitude or GRE tests are given by

(a)	**(b)**
house : hut	**(b) water : dam**
tree : _____	**_____ : battery**
(c)	(d)
green : go	# 0 *
red : _____	0 *
	* # 0

Examples of more abstract analogies are the planetary system or atomic model noted above, the proof of a theorem based on a similar known proof, solving a new problem from knowledge of an old familiar problem solving technique, learning to play the card game of bridge from a knowledge of hearts, or producing a new algorithm for a program using previously learned programming examples and concepts.

Although applications of analogical methods have received less attention in AI than other methods, some important results have been published including (Burstein, 1986, Carbonell, 1983 and 1986, Greiner, 1988, Kling, 1971, McDermott, 1979, and Winston, 1980). Researchers in related fields have also made important contributions: cognitive science (Gentner, 1983), and psychology (Rumelhart and Norman, 1981). Several of the researchers in AI have produced working programs based on their models of the analogical process. We examine some of these models in the following section. In the remainder of this section we investigate the analogical reasoning process in some detail.

The Analogical Reasoning Process

Analogical reasoning was briefly described in Chapter 4 as a nonvalid form of inference. This follows from the fact that conclusions based on analogies are not necessarily logical entailments of the source knowledge. Analogy is a form of plausible reasoning. It is based on previously tested knowledge that bears a strong resemblance to the current situation.

There are five essential steps involved in the analogical learning process. A central step is the mapping process which is illustrated in Figure 20.1 and described below.

Analogical Learning Process

1. Analogue recognition: A new problem or situation is encountered and recognized as being similar to a previously encountered situation

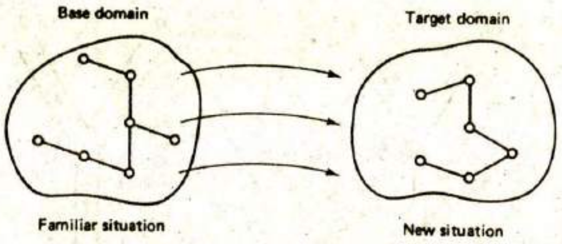


Figure 20.1 The analogical mapping process.

2. Access and recall: The similarity of the new problem to previously experienced ones serves as an index with which to access and recall one or more candidate experiences (analogues).
3. Selection and mapping: Relevant parts of the recalled experiences are selected for their similarities and mapped from the base to the target domain.
4. Extending the mapped experience: The newly mapped analogues are modified and extended to fit the target domain situation.
5. Validation and generalization: The newly formulated solution is validated for its applicability through some form of trial process (such as theorem provers or simulation). If the validation is supported, a generalized solution is formed which accounts for both the old and the new situations.

Implementation of the above process requires that several important issues be addressed. These issues are recurrent ones that appeared several times in other problems.

First, there is the question of representation. For analogical reasoning, it is desirable that knowledge be stored in a form that is easily accessed, retrieved, and evaluated for possible use as a candidate solution. This implies that self-contained, interrelated pieces of knowledge comprising a particular situation, episode, proof, plan, concept, and other unit of knowledge should be indexed and stored for simultaneous recall. Object attributes and relationships should be bound together with identifiers and other descriptive information for ease of use.

Second, it is desirable that an appropriate similarity measure be used to insure that only candidate analogues bearing a strong similarity to the new situation be considered.

Third, it is important that the mapping process from base to target domain be flexible enough to capture and import only the appropriate parts of the descriptions. The mapping should be able to transform objects, attributes, and predicates into corresponding ones which best meet the requirements of the target domain situation. For a general analogical system, this means that the transformation process should be dynamically adaptable to map between different domains and at different levels of abstraction.

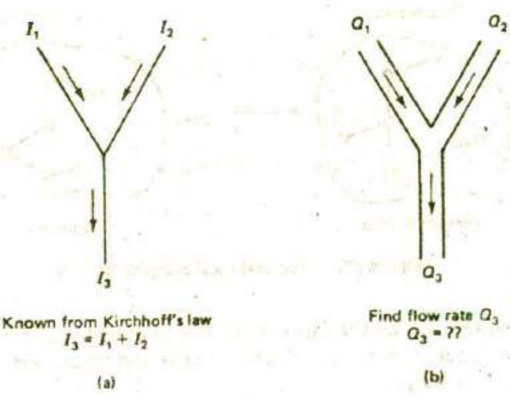


Figure 20.2 Analogical problem solving example.

Next, the newly created solution must be tested for its suitability. This test can be as informal as a simulated solution trial in the target domain or as formal as a deductive test of logical entailment.

Finally, having found an analogy and tested it successfully, the resultant episode should be generalized if possible and then summarized, encoded, indexed and stored for subsequent use in reasoning or learning.

A simple example will help to illustrate this process. Suppose we are given the problem of determining the flow rate of a fluid from a simple Y junction of pipes (Figure 20.2b). We are asked to determine the value of Q_3 given only knowledge of the flow rates Q_1 and Q_2. A description of this unknown problem reminds us of a similar known problem, that of finding the flow rate of electrical current in a circuit junction. We recall the solution method to the electrical problem as being that based on Kirchhoff's current flow law, namely, that the sum of the currents at a junction is zero. We use this knowledge in an attempt to solve the hydraulic flow problem using the same principles; that is, we map the electrical flow solution to the hydraulic flow problem domain. This requires that corresponding objects, attributes and relations be suitably mapped from the electrical to the hydraulic domain. We then test the conjectured solution in some way.

In the reminding process, we may alternatively be given a direct hint from a teacher that the hydraulic flow problem is like the electrical flow problem. Otherwise, we must infer this likeness in some way and proceed with the conjecture that they are alike, justifying the likeness on the basis of the consistency of nature or some other means.

Next, we examine some representative examples of analogical learning systems.

20.3 EXAMPLES OF ANALOGICAL LEARNING SYSTEMS

Winston's System

Patrick Winston (1980) developed programs that reason about relationships, motives, and consequent actions that occur among people. Using relationships and acts of actors in one story (such as *Macbeth*) the program was able to demonstrate that analogous results occurred in different stories (such as *Hamlet*) when there were similarities among the relationships and motives of the second group of characters. The programs could also learn through the analogical reasoning process. For example, when a teacher declared that voltage, current, and resistance relationships were like those of water pressure, flow, and pipe resistance, the system was able to learn basic results in electrical circuits and related laws such as Ohm's law (the opposite of the learning problem described above).

The analogical mapping and learning process for this example is illustrated in Figure 20.3. The items in the figure labeled as voltage-value-3, current-value-3, and resistance-value-3 represent specific values of voltage, current, and resistance, respectively.

The important features of Winston's system can be summarized as follows:

1. Knowledge representation: Winston's system used frame structures as part of the Frame Representation Language (FRL) developed by Roberts and Goldstein (1977). Slots within the frames were given special meanings, such as AKO, appears-in, and the like. Individual frames were linked together in a network for easy access to related items.

2. Recall of analogous situations: When presented with a current situation, candidate analogues were retrieved from memory using an hierarchical indexing scheme. This was accomplished by storing a situations (frame) name in the slots of all object frames that appeared in the situation. For example, in the Cinderella story, the prince is one of the central parts. Therefore, prince would be used as a node in a hierarchical tree structure with sublinks

$$\text{prince} \rightarrow \text{AKO-man} \rightarrow \text{AKO-person}$$

and the Cinderella label CI would be stored in an appears-in slot of the frames belonging to the prince, the man, and the person. These slots were always searched as part of the recall reminding process when looking for candidate analogues.

3. Similarity matching: In selecting the best of the known situations during the reminding process described above, a similarity matching score is computed for each of the recalled candidates. A score is computed for all slot pairings between two frames, and the pairing having the highest score is selected as the proper analogue. The scoring takes into account the number of values that match between slots as

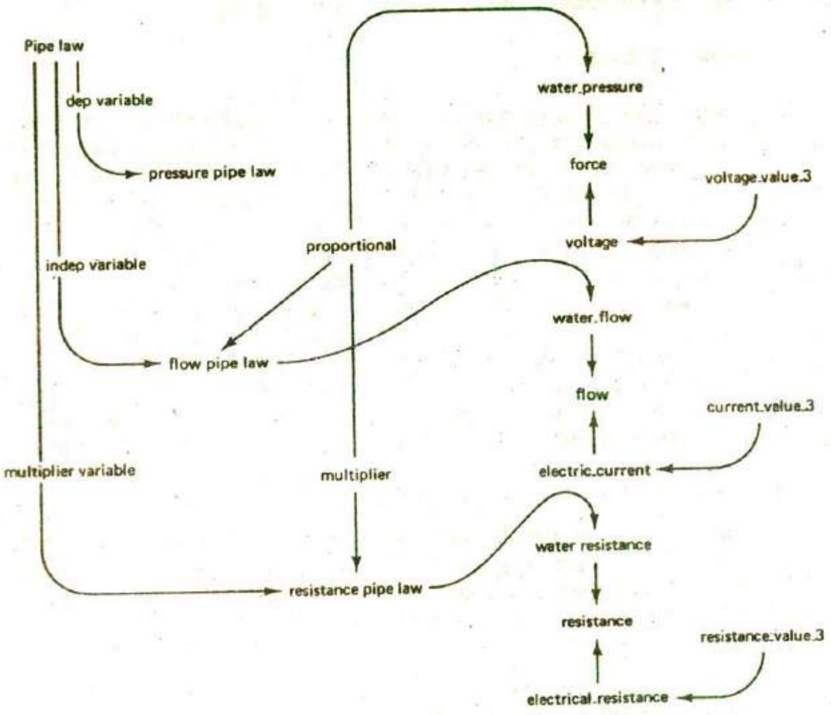
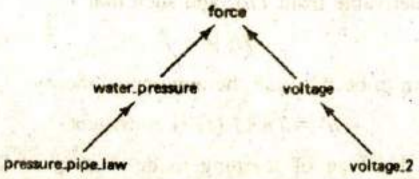


Figure 20.3 Learning electrical laws from hydraulics.

well as matching relationships found between like parts having causal relations as noted in comment fields.

4. Mapping base-to-target situations: The base-to-target analogue mapping process used in this system depends on the similarity of parts between base and target domains and role links that can be established between the two. For example, when both base and target situations share the same domain, such as water-flow in pipes, parts between the two are easily matched for equality. A specific pipe, pipe1, is matched with a general pipe, pipe#, specific water-flow, with general flow, and so forth. The relationships from the general case are then mapped directly to the specific case without change.

In cases where base and target are different domains, the mapping is more difficult as parts, in general, will differ. Before mapping is attempted, links are established between corresponding parts. For example, if the two domains are water-flow and electricity-flow (it is known the two are alike), determining the electrical resistance in a circuit is achieved by mapping the pipe-water-flow laws to the electrical domain. Before mapping, a link between the pressure and voltage laws and the water-flow and current-flow laws is established. The link is determined from a hierarchical tree connecting the two parts, where voltage-2 is a specific voltage.



In all of the above components, matching plays a dominant role. First, in recall, matching is used to determine candidate situations. Second, matching is used for scoring to select the best analogue. Finally, matching is used to determine how parts and relationships are mapped from base to target domains.

Greiner's NLAG System

Russell Greiner (1988) developed an analogical learning system he called NLAG using Stanford University's MRS logic development language. The system requires three inputs and produces a single output. The inputs are an impoverished theory (a theory lacking in knowledge with which to solve a given problem), an analogical hint, and the given problem for solution. The output from the system is a solution conjecture for the problem based on the analogical hint.

Using Greiner's notation, the process can be described as follows. The analogical inference process described here is represented by the operator $\mid\!\sim$. This operator takes three inputs.

1. A finite collection of consistent propositions (rules, facts, and the like) called a theory (*Th*).
2. An analogical hint about the source and target analogues *A* and *B* respectively, written $A \sim B$ (*A* is like *B*). Here $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ are sets of arbitrary formulae or knowledge related to some problem situations.
3. A problem to be solved in the target domain (target problem) denoted as *PT*.

The output from the system is a set of new propositions or conjectures $\phi(A)$ related to the set *B* that can be used to solve *PT*.

The above process can be summarized as follows:

$$Th, A \sim B \models_{PT} \phi(A)$$

The NLAG system was designed to perform useful, analogical learning. We define the *learning* part here as the process of determining or creating one or more formulas f which are not initially in the deductive closure of Th, that is, such that

$$Th \not\models f \text{ holds,}$$

(so f is not initially derivable from Th), and such that

$$Th \not\models \sim f,$$

that is, f is not known to be false; so the augmented theory

$$Th' = Th \cup \{f\} \text{ is consistent.}$$

We specialize this type of learning to be *analogical learning* by requiring that the process use the operator $\mid\!\sim$ and have as inputs both the theory Th as well as the hint $A \sim B$. Also, the new formulae f should be about the target analogue A ($f = \phi(A)$ where ϕ is some set of arbitrary formulae). The source analogue B should also satisfy the analogy formulae. Thus, we must also have $Th \models \phi(B)$.

Finally, we specialize the definition further to *useful analogical learning* by requiring that the conjectured formulae $\phi(A)$ returned by the system be limited to those formulae that are useful in the sense that the ϕ, together with Th, can solve PT, that is

$$Th \cup \{\phi(A)\} \models PT.$$

This definition of useful analogical learning is summarized in Figure 20.4 where we have named the conditions described in the above definition as unknown, consistent, common, and useful.

These notions are illustrated in terms of our earlier example for the hydraulics or electrical flow problem (Figure 20.2). Thus, B here corresponds to the known theory Th that given an electrical Y junction as in Figure 20.2a, then $I_3 = I_1 + I_2$. The problem PT is to find Q_3, the fluid flow rate in a similar Y junction of pipes when $Q_1$and Q_2 are known. The analogical hint, $A \sim B$, is that the hydraulics flow problem is like the electrical current flow problem. The useful formula needed to solve the problem is of course $Q_3 = Q_1 + Q_2$. A more complex example would, of course, require more than a single formula.

Since many analogies may satisfy the theory and the analogical hint, it is necessary to restrict the analogies considered to those which are useful. For this, NLAG uses heuristics to select only those formulae which are likely to be useful.

One heuristic used by the system is based on the idea that relationships in

$$Th, A \sim B \mid\!\sim_{PT} \phi(A)$$

where	UNKNOWN:	$Th \not\models \phi(A)$
	CONSISTENT:	$Th \not\models \sim\phi(A)$
	COMMON:	$Th \models \phi(B)$
	USEFUL:	$Th \cup \{\phi(A)\} \models_{PT}$

Figure 20.4 Summary of useful analogical inference.

one domain should hold in other, similar domains (in two domains governed by physical laws). In our example, this is the law related to zero flow rates into a junction and the corresponding reusable zero-sum formula for general, physical flows. Thus, only those uninstantiated abstract formulas found in the base analogue would be permitted as conjectures in the target analogue solution. Furthermore, this same heuristic requires that the selected formulae be atomic; formulae of the form $f(a_1, a_2, \ldots, a_k)$ are permitted, whereas formulae containing multiple conjunctive or disjunctive terms such as

$$f_1(a,b) \,\&\, f_2(b,c) \vee f_3(a,c,d,e)$$

are not permitted. These formulae must also satisfy the usefulness condition (Figure 20.4).

A heuristic which helps to further prune the abstract solutions described above uses the target domain problem *PT*, to suggest a related query in the source domain. The corresponding source domain query *PS* is then used in turn to select only those formulae which are both abstractions (as selected above) and are also relevant. For example, the target query *PT* = "Find the flowrate in the given pipe structure" is used to find the analogous source domain query *PS* = "Find the current in the given electrical structure." This query is then used to determine which facts are used to solve *PS*. These facts are then used as a guide in selecting facts (formulae) for the target domain to solve *PT*.

Other system heuristics require that formulae must all come from the same domain, that more general abstractions be preferred over less general ones, and that for a given abstraction, instances which require the fewest conjectures be chosen. In general, these heuristics are all based on choosing only enough new information about the target analogue to solve *PT* and then stop.

Carbonell's Systems

Carbonell developed two analogical systems for problem solving, each based on a different perception of the analogical process (1983 and 1986). The first system was based on what he termed transformational analogy and the second on derivational analogy. The major differences between the two methods lie in the amount of details remembered and stored for situations such as problem solution traces and the methods used in the base-to-target domain transformational process.

Both methods used a knowledge representation and memory indexing and recall scheme similar to the memory organization packets or MOPs of Roger Schank (Chapter 11). Both methods also essentially followed the five-step analogical learning process outlined in the previous section. The main differences between the two methods can be summarized as follows:

Transformational Analogy. Problem solving with this approach is based on the use of means-end analysis (MEA) as described in Chapter 9. Known problem solutions are indexed and stored for later retrieval. The solutions are stored as an initial state, a goal state, and a sequence of actions (operators) which, when applied to the initial and intermediate states, result in a transformation to the goal state. When a new problem is encountered, it is matched against potentially relevant known ones using a suitable similarity measurement. The partial match producing the highest similarity measure is transformed to satisfy the requirements of the new problem. In finding a solution to the new problem using the known mapped solution, it is often necessary to disturb some states to find operators which reduced the current-to-goal state differences. With this method, the focus is on the sequence of actions in a given solution, and not on the derivation process itself.

Derivational Analogy. This approach requires the storage of much more information than the transformational approach. It is intended that the analogies here be made more on the basis of the complete reasoning process than on the sequence of solution states for the past problem solutions. In solving newly encountered problems, a plan is formulated using the complete solution trace of the analogue. This plan includes a list of all subgoals derived, alternative solution paths generated and those rejected, each step of the solution process and the decisions made at each step, pointers to knowledge which proved to be useful in the solution, and detailed failure information when a solution cannot be found. This plan is modified to fit the new problem domain. If it cannot produce a solution, other methods are applied such as MEA.

Both of the systems developed by Carbonell are useful as analogical research tools. And both worked reasonably well in more than one domain.

20.4 EXPLANATION-BASED LEARNING

One of the most active areas of learning research to emerge during the early 1980s is explanation-based learning (also known as explanation-based generalization). This is essentially a form of deductive generalization. It has been successfully demonstrated for the general area of concept learning.

In EBL, four kinds of information must be known to the learner in advance.

1. A formal statement of the goal concept to be learned,
2. At least one positive training example of the concept,
3. Domain theory, which relates to the concept and the training example, and
4. Criteria for concept operationality.

The notion of operationality in item 4 requires some explanation. We say a procedure is operational if, for some agent and task, the procedure can be applied by the agent to solve the task. Thus, if the concept to be learned is one related to the recognition of pencils, operational criteria in this case would be given as a structural definition which could serve to uniquely identify pencils. If the concept is related to the function or use of pencils, the operational criteria would be given as a functional definition.

The objective of EBL is to formulate a generalized explanation of the goal concept. The explanation should prove how the training example satisfies the concept definition in terms of the domain theory.

The EBL method constructs an explanation of the concept based on the given example in terms of the domain theory. The explanation is a deductive proof of how the example satisfies the goal concept definition. Once an explanation has been developed, it is generalized by regressing the goal concept through the explanation structure. We illustrate these ideas with an example given by Mitchell, Keller, and Kedar-Cabelli (1986).

The concept to be learned in this example is the conditions under which it is safe to stack one item on top of another item. The training example is given as a description of a pair of objects; one object is a box which is stacked on top of the other object, an end table. The domain knowledge needed by the system consists of some structural attributes and relationships between physical objects such as volume, density, weight, lighter-than, fragile, and so on. The operational criteria in this case requires that the concept definition be expressed in terms of the predicates used to describe the example. The objective of the learning process here is to develop a generalization of the training example that is a sufficient concept definition for the goal concept which satisfies the operationality criteria. These definitions are summarized in Figure 20.5.

In carrying out the EBL method, the system must first construct an explanation of how the training example satisfies the goal concept. This requires that it select those attributes and relations which are relevant and which satisfy the operational criteria. From the goal concept definition it is seen that it is safe-to-stack an object x on an object y if and only if either y is not fragile or x is lighter than y. A search through the domain theory for the predicate FRAGILE fails; therefore, a search is then made for the predicate LIGHTER. This predicate is found in the rule

$$\text{WEIGHT}(p_1,v_1)\ \&\ \text{WEIGHT}(p_2,w_2)\ \&\ \text{LESS}(w_1,w_2) \rightarrow \text{LIGHTER}(p_1,p_2)$$

Matching obj1 to x (to p_1 in the above rule) and obj2 to y (to p_2 in the above rule) we can form the following partial explanation for safe-to-stack.

$$\begin{tikzcd}
& \text{SAFE-TO-STACK(obj1,obj2)} \arrow[d] & \\
& \text{LIGHTER(obj1,obj2)} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{WEIGHT}(\text{obj1},w_1) & \text{LESS}(w_1,w_2) & \text{WEIGHT}(\text{obj2},w_2)
\end{tikzcd}$$

Continuing with the explanation process, we find rules relating the weights with volume and density and to an endtable. After instantiation of these terms, we

Known:

Concept Definition:

Pairs of objects <x,y> are SAFE-TO-STACK when
SAFE-TO-STACK(x,y)↔ ~FRAGILE(y) V LIGHTER(x,y).

Training Example:

ON(obj1,obj2)
ISA(obj1,box)
ISA(obj2,endtable)
COLOR(obj1,red)
COLOR(obj2,blue)
VOLUME(obj1,1)
DENSITY(obj1,0.1)
. . .

Domain Theory:

VOLUME(p_1,v_1) & DENSITY(p_1,d_1) → WEIGHT(p_1,v_1*d_1)
WEIGHT(p_1,w_1) & WEIGHT(p_2,w_2) & LESS(w_1,w_2) →
LIGHTER(p_1,p_2)

ISA(p_1,endtable) → WEIGHT(p_1,5) (i.e. a default)
LESS(0.1,5)
. . .

Operationality Criteria:

The learned concept should be expressed in terms of the same predicates used to describe the example, i.e. COLOR, DENSITY, VOLUME, etc. or simple predicates from the domain theory (e.g. LESS).

Determine:

A generalization of the training example that is a sufficient goal concept definition which satisfies the operationality criteria.

Figure 20.5 Safe-to-stack example for EBL.

are led to the complete explanation tree structure given below in which the root terms are seen to satisfy the operationality criteria.

$$\begin{tikzcd}
 & & \text{SAFE-TO-STACK(obj1,obj2)} \arrow[d] & \\
 & & \text{LIGHTER(obj1,obj2)} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \text{WEIGHT(obj1,0.1)} \arrow[dl] \arrow[dr] & \text{LESS(0.1,5)} & \text{WEIGHT(obj2,5)} \arrow[d] \\
\text{VOLUME(obj1,1)} & & \text{DENSITY(obj1,0.1)} & \text{ISA(obj1,endtable)}
\end{tikzcd}$$

The next step in developing the final target concept is to generalize the above structure by regressing or back-propagating formulae through rules in the above structure step by step, beginning with the top expression SAFE-TO-STACK and regressing SAFE-TO-STACK(x,y) through the goal rule (the ~ FRAGILE disjunct is omitted since it was not used). LIGHTER(p_1,p_2) → SAFE-TO-STACK(p_1,p_2) yields the term LIGHTER(x,y) as a sufficient condition for inferring SAFE-TO-STACK(x,y). In a similar way LIGHTER(x,y) is regressed back through the next step to yield WEIGHT(x,w_1) & WEIGHT(y,w_2) & LESS(w_1,w_2). This expression is then regressed through the final steps of the explanation structure to yield the following generalized, operational, definition of the safe-to-stack concept.

```
VOLUME(x,v1)
& DENSITY(x,d1)
& LESS(v1*d1,5)
& ISA(y,endtable) → SAFE-TO-STACK(x,y)
```

The complete regression process is summarized in Figure 20.6 where the underlined expressions are the results of the regression steps and the substitutions are as noted within the braces.

In summary, the process described above produces a justified generalization of a single training example as a learned concept. It does this in a two step process. The first step creates an explanation that contains only relevent predicates. The second step uses the explanation structure to establish constraints on the predicate values that are sufficient for the explanation to apply in general. This differs from inductive learning in that a single training example is adequate to learn a valid description of the concept. Of course there is a trade-off here. The EBL method also requires appropriate domain knowledge as well as a definition of the target concept.

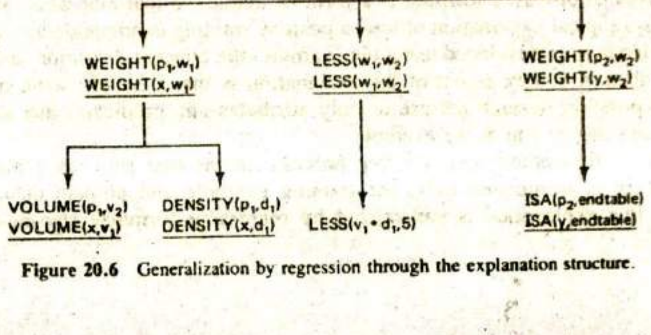


Figure 20.6 Generalization by regression through the explanation structure.

There is an apparent paradox in the EBL method in that it may appear that no actual learning takes place since the system must be given a definition of the very concept to be learned! The answer to this dilemma is that a broader, generalized, and more useable definition is being learned. With the EBL method, existing knowledge is being transformed to a more useful form. And, the learned concept applies to a broader class than the supplied definition. This newly learned concept is a valid definition since it has been logically justified through the explanation process. The same claim cannot be made of other nondeductive learning techniques like inductive or analogical learning.

The notion of operationality used in EBL systems should depend on the purpose of the learning system. As such, it should be treated as a dynamic property. For example, a robot may need to learn the concept of a pencil in order to recognize pencils. In this case, operationality should be interpreted in terms of the structural properties of the pencil. On the other hand, if the purpose of learning the pencil concept relates to design, the robot would be better served with a functional definition of a pencil. Keller (1988) discusses the operationality problem and its application in a program called MetaLEX (a successor to LEX described in Chapter 19).

20.5 SUMMARY

We have described two of the more promising approaches to machine learning, the analogical and explanation-based learning paradigms. These methods, unlike similarity-based methods, are capable of creating new knowledge from a single training example. Both methods offer great potential as autonomous learning methods.

Learning by analogy requires that similar, known experiences be available for use in explaining or solving newly encountered experiences. The complete process can be described in five steps: (1) a newly encountered situation serves as a reminder of a known situation, (2) the most relevent of the reminded situations are accessed and recalled, (3) the appropriate parts of the recalled analogues are mapped from the base domain to the target domain, (4) the mapped situation or solution is extended to fit the current problem, and (5) the extended solution is tested, generalized, and stored for subsequent recall.

A number of analogical research learning systems have been developed. Four representative systems have been described in this chapter.

Explanation-based learning is a form of deductive learning where the learner develops a logical explanation of how a positive training example defines a concept. The explanation is developed using the example, the concept definition, and relevant domain theory. A key aspect of the explanation is that it satisfy some operational criteria, possibly through the use of only attributes and predicates that are used in the domain theory and/or the example.

The EBL method is a two step process. In the first step, an explanation of the concept is formulated using the training example and domain theory. In the second, this explanation is generalized by regressing formulae step-by-step back through rules in the explanation structure. The final concept, if successful, is a definition which both satisfies the operationality criteria and is also a valid definition of the concept.

EXERCISES

20.1. Describe two examples of analogical learning you have experienced recently.

20.2. Why is it that things similar in some ways tend to be similar in other ways?

20.3. Consult a good dictionary and determine the differences between the definitions of analogies, metaphors, and similes.

20.4. Make up three new analogies like the examples given in Section 20.2.

20.5. Relate each of the five steps followed in analogical learning to the following example: Riding a motorcycle is like riding a bicycle with an engine in it.

20.6. Compare the analogical system of Winston to that of Greiner. In what ways do they differ?

20.7. What appears to be more important in mapping from base to target domain, object attributes, object relationships, or both? Give examples to support your conclusions.

20.8. What are the main differences between Carbonell's transformational and derivational systems?

20.8. Define operationality as it applies to explanation-based learning and give an example of it as applied to some task.

20.10. Explain why each of the four kinds of information (concept definition, positive training example, domain theory, and operational criteria) is needed in EBL.

20.11. If the end table in the Safe-to-Stack example had not been given a default weight value, what additional steps would be required in the explanation to complete the tree structure?

20.12. What is the purpose of the regression process in EBL?

20.13. Work out a complete explanation for the concept safe to cross the street. This requires domain theory about traffic lights, and traffic, a positive example of a safe crossing, and operational criteria.

20.14. The learning methods described in Part V have mostly been single paradigm methods. yet we undoubtedly use combined learning for much of our knowledge learning. Describe how analogical learning could be combined with EBL as well as inductive learning to provide a more comprehensive form of learning.