

- a. Learning entails modifications to the domain-specific knowledge portion of the knowledge base, possibly by adding or modifying production rules - in a symbolic system - or by modifying the weights of neural net connections - in a connectionist system. This distinguishes learning from ordinary computation, which alters only the problem state portion of the knowledge base.
 - b. Learning occurs under the control of the engine, rather than by manipulation of the knowledge base by a human programmer. This distinguishes learning from ordinary program maintenance.
2. Adding learning to a system like this requires adding several elements to the architecture.
- a. The engine is divided into two major components - a performance element that solves problems (as always) and a learning element that adjusts the knowledge base in light of how well the performance element did.
 - b. A critic evaluates the system's performance and gives feedback to the learning element.
 - c. A problem generator poses problems for the system to solve.
3. Learning, as we have defined it, can occur in several ways. These differ primarily in the nature of the critic and the problem generator.
- a. The system can be taught by a human expert, using carefully chosen training pairs consisting of an example problem and the correct response. That is, a human being serves as problem generator and critic.
 - i. This approach has been successfully used for teaching systems to recognize different objects or patterns (such as arches in the blocks world.)
 - ii. Under this approach, the human expert is in control in that he chooses the sequence of examples to be presented to the system.
 - iii. However, this differs from conventional program maintenance in that the human expert does not directly modify the knowledge base; rather, the engine modifies the knowledge base in response to the data presented to it.
 - iv. This is analogous to a structured course such as this one, in which I, as the professor, plan a series of learning activities through which (I hope) you will master the material, and provide you access to the correct answers after you have attempted the problem.

This approach to learning is called SUPERVISED LEARNING.
 - b. The system can perform in its intended environment, with feedback as to whether or not its performance is correct. In this case, as compared to the preceding case, the feedback is boolean - whether or not the system did the right thing. The system is not given the correct answer if it did the wrong thing. That is, the system's environment serves as problem generator and critic.

- i. This approach has been successfully for teaching systems to play two-player games. (The only feedback the system receives is whether or not it won the game.)
 - ii. This is called REINFORCEMENT LEARNING.
 - iii. It may appear that the example of teaching a neural network to be an and gate that we did earlier in the course falls into this category, since the feedback given to the net was "you got it wrong" when it gave the wrong response to an input pattern. However, since with a boolean function there are only two possible answers, saying "you got it wrong" also implicitly tells the system what the right answer is. (In fact, the example we did earlier is really supervised learning, because we cycled through a fixed set of training examples and the net was wired to learn to give the correct responses to each.)
 - c. The system can learn by experimentation - either by actually conducting physical experiments or (at the present state of the art) by conducting "thought experiments". That is, the system itself serves as problem generator and critic.
 - i. Among humans, the ability to learn in this way distinguishes an expert in a field from a neophyte. This approach to learning is exemplified by the theses required in many graduate programs (and some undergraduate programs) and by things like the senior project required here at Gordon.
 - ii. However, this kind of learning is not limited to human experts. In fact, for a new-born child this is the first kind of learning he does, as he gradually discovers his bodily faculties and how to control them!
 - iii. This is called UNSUPERVISED LEARNING.
 - d. No one of these three methods is best-each has its appropriate uses.
- E. Learning is of interest not only in its own right, but also to support other AI applications. For example, consider the possible role of learning in expert systems:
1. One of the key bottlenecks in producing expert systems is the problem of transferring knowledge to the system from reference works or human experts.
 2. This problem could be greatly reduced if automated learning of the first or second general types could be used.
 3. It would be even nicer if expert systems, once equipped with a basic set of rules, could modify themselves to improve their performance, thus turning failures into learning experiences. (This is, of course, exactly what humans do.)
- F. Actually, learning systems come in three rather distinct "flavors".
1. Symbolic learning - in which the system learns rules, or parameters of rules. We consider this today and next class.

2. Connectionist learning - in which the system learns connection weights in a neural network. We consider this in the next series of lectures.
 3. Genetic learning - a very different approach, modelled on natural selection. We will consider this last.
- G. Because human learning, in general, is not well understood, it should not be surprising that machine learning is still very much a research frontier, with very different approaches used to solve different problems. The literature on this topic is extensive.

At this point, we will look at three examples of symbolic approaches. Each involves a system of historical importance in the development of symbolic learning techniques.

1. A numerical approach, based on "tuning" the parameters of a set of rules without altering the structure of the rules. This will be an illustration of reinforcement learning.
2. A structural approach which alters and expands the rule structure itself, by refining an existing set of knowledge categories and rules. This will illustrate supervised learning.
3. An exploratory approach which actually generates new knowledge categories and rules. This will illustrate unsupervised learning.

II. An Example of Reinforcement Learning Using A Numerical Approach

-- -- -----

- A. In a numerical approach to learning, a set of rules is written with numerical parameters that can be altered by the system to improve its performance.
1. Much of the early work in this area was done in conjunction with games playing programs, where numerical parameters can be used to control the static evaluation function and search tree pruning heuristics.
 2. An interesting example of this is one of the earliest AI programs: Arthur Samuel's checkers-playing program. Because it was one of the earliest AI programs, and because the personal story surrounding it is quite interesting, I am going to read at length from Pamela McCorduck's discussion of it in *Machines Who Think*.

READ MCCORDUCK PP. 148-153

- B. To illustrate how learning of this kind can be accomplished, we begin with an extremely simple (actually trivial) illustration of this approach: the game of Hexapawn.
1. READ LOGSDON PP 234-237 MIDDLE - PROJECT FIGURES 65, 66
 2. DEMO SEVERAL GAMES WITH MATCHBOX SET

III. An Example of Supervised Learning

--- -- -----

- A. Both of the previous learning examples were limited by the fact the basic STRUCTURE of the solution was developed by a human programmer, with only a few numerical coefficients to be learned (or boolean ones in the case of Hexapawn.) More sophisticated learning systems must be able to learn at the structural level as well.
- B. Much of the classic work in learning of the this sort has been done in the realm of class recognition: teaching programs to recognize objects which are and are not members of some class of objects, or to recognize to which of several classes they belong.
 - 1. That is, we have a (presumably large) set of objects.
 - a. From this, we select two non-intersecting sets: a training set, and a test set.
 - b. We teach the system to correctly classify members of the training set.
 - c. We hope that it will correctly classify members of the test set, based on what it learned from the training set.
 - d. Of course, this presumes that the classification scheme has some learnable properties - e.g., as the book discusses, the concept of "ball" is learnable by being presented with examples of balls and non-balls; but the concept "member of a set of randomly-selected objects" is not learnable, because the only way to correctly classify all of them is to see all of them!
 - 2. The early work that is often cited as an example of this is Winston's program that was able to learn classification rules for structures in a blocks rule. In particular, the example we will use is the problem of teaching a program to recognize an "arch" in the blocks world. (We will look at this shortly).
 - 3. While blocks-world learning of this sort is not particularly useful, in this case the same techniques have been transferred to more significant problems. In particular, Winston cites the work of Michalski, who developed a program that
 - "learned criteria for recognizing more than a dozen soybean diseases, producing results superior to human specialists. Since a considerable fraction of the world's population relies on soybeans for survival, to learn to recognize soybean diseases is do something important."(pp 385-386)
 - 4. The basic idea in learning programs of this sort is this: a human teacher presents the program with a series of descriptions of objects, some of which belong to the class in question and some of which do not. (The program is told which is which.)
 - a. Each description consists of a collection of features.

- b. The program eventually must learn that some features are necessary to an object belonging to the class; others are typically true of objects of the class; others rule out the object as being a member of the class (they must not be present); and others are irrelevant.
- c. The learning process goes most effectively if the examples and the order in which they are presented is well chosen.
 - i. The first example should be a typical example of the class.
 - ii. Examples that are not members of the class should be NEAR MISSES: objects that differ from class members in one significant way. (From these the program learns either that a certain feature that is present in the class members but not in the miss must be present in objects of the class, or that a feature that is absent in class members but present in the near miss must not be present in objects of the class.)
 - iii. Some examples of class members will be EXTREME examples: objects that just barely qualify for membership in the class. From these the program learns that certain features present in many class examples are not strictly necessary for class membership.)
- d. Example: PROJECT WINSTON FIGURE 11-1
 - i. The first example is a typical member of the class "arch".
 - ii. The second example is a near miss that teaches that the fact that two objects support the third is a necessary feature of arches.
 - iii. The third example is a near miss that teaches that the two support objects touching each other is a feature that must not be present in arches.
 - iv. The fourth example teaches that the fact that the top object is rectangular is not a necessary feature of arches.

5. Winston lists a number of heuristics that a learning program of this sort can use.

PROJECT: P. 392 (BOTTOM)

- C. One of the later pieces of work building on Winston's original work is the concept of VERSION SPACES. This is a formalization of these heuristic rules.
 - 1. The basic idea is this: at any point in the learning process, there are many rules that could potentially be used to discriminate the objects presented thus far that belong to the class from objects presented thus far that do not belong. We assume that these rules take the form of a conjunction of predicates which an object must satisfy to be regarded as a member of the class.

a. Example: Tanimoto figure 8.2 (PROJECT)

Suppose the program has only been presented with a [an example] and c [a near miss] so far (not b or d). Then all of the following appear to be potential candidates for discriminating arches from non-arches:

Hatched(b2) R1

-Adj(b1, b3) R2

$On(b2, b1) \cap On(b2, b3) \cap \neg Adj(b1, b3)$ R3

$On(b2, b1) \cap On(b2, b3) \cap \neg Adj(b1, b3) \cap Hatched(b2)$ R4

b. Of course, the third of these is the correct rule (leaving out any requirements about object shape). We would expect the program to converge on this eventually.

2. The set of all rules that successfully discriminate examples from near misses at a given point in time is called a version space. As each new example or miss is presented, a new (and smaller) version space is created. (Actually, the version space could be the same size if the example is poorly chosen and the program can learn nothing from it.)

3. It is possible to order the rules of a version space from least general to most general. We say that a rule $R \leq S$ iff whenever R says an object is a member of the class then S does to.

i.e. $R \leq S \iff (\forall x) [R(x) \rightarrow S(x)]$

a. Example: in the above set of rules,

$R3 \leq R2$

$R4 \leq R1, R4 \leq R2, R4 \leq R3$

b. This allows us to identify one or more LEAST GENERAL rules. The least general rules are those rules R for which no other rule S exists such that $S \leq R$.

(In the above, $R4$ is the only rule meeting this criterion.)

c. We can also identify the MOST GENERAL rules as those rules R for which no other rule S exists such that $R \leq S$.

(In the above, both $R1$ and $R2$ meet this criterion - note that there is not a unique most general rule. [Actually, there may not be a unique least general rule either - though in this case there happens to be])

d. The entire version space can be characterized by listing the set of most general rules and the set of least general rules, omitting all the others (which now lie between these two extreme sets.)

Example: after presenting (a) and (c), the version space could be characterized by:

Least General: $\{On(b2,b1) \cap On(b2,b3) \cap \neg Adj(b1,b3) \cap Hatched(b2)\}$
Most General: $\{ Hatched(b2), \neg Adj(b1, b3) \}$

4. A new example allows us to refine the version space as follows:

- a. If the example is a member of the class, then we can drop from the previous extreme rules any predicates which the new example does not satisfy. (In many cases, only the least general rule will have to change.)

Example: if after being presented with the first and third objects, the system was now presented with the second, it would be necessary to drop the Hatched(b2) predicate from the least general rule to get $On(b2, b1) \wedge On(b2, b3) \wedge \neg Adj(b1, b3)$.

We would also have to drop the entire most general rule Hatched(b2).

- b. If the example is not a member of the class, then we can add conjuncts to the least and most general rules that rule out the near-miss just considered. (In many cases, only the most general rule would have to change, and the needed conjuncts can be taken from the least general rule.)

Example: if after being presented with the first and third objects, the system was now presented with the fourth, it would be necessary to add conjuncts to both of the most general rules $\neg Adj(b1, b3)$ and Hatched(b2). In each case, the conjunct that we would need to take from the least general rule is $On(b2, b3)$; this is the only one in the least general rule that is not satisfied by the near miss.

Our most general rules would now become:

$\neg Adj(b1, b3) \cap On(b2, b3)$
 $Hatched(b2) \cap On(b2, b3)$

5. The entire process of learning arch rules is developed on Tanimoto page 295.

PROJECT

NOTE WELL THE ORDER IN WHICH THE EXAMPLES ARE PRESENTED!

6. In the examples above, we have assumed that the set of examples to present to the program was chosen by the human teacher. It is also possible (and in many cases desirable) to build a learning system where the set of examples presented is chosen randomly. Here, the role of the human "teacher" is simply to provide the correct classification of the example which the program is to learn.

- D. Another approach to supervised symbolic learning involves inducing DECISION TREES from large pools of data.
1. An example: data on loan applicants
PROJECT Luger table 10.1
 2. Some examples of decision trees that correctly classify this data:
PROJECT Luger Figures 10.13, 10.14
 3. The book discusses an algorithm known as ID3 that "learns" a decision tree from a set of raw data. It uses information content considerations to select which piece of information to make the root of the tree - e.g. Luger shows that, for this set of data, ID3 would produce the second tree rather than the first.
- E. Many practical applications of supervised approaches to learning have the following general form:
1. We have a large - possibly infinite - space of possible instances which we would like our program to classify correctly (e.g. arch or not arch - or perhaps some more complex classification involving many possibilities.)
 2. We can teach our program by selecting a subset of all the possible instances called a TRAINING SET, giving the program the instance and the correct answer, and allowing it to develop a classification rules that gives the correct answer for every instance in the training set. Our hope is that the program will learn a classification rule that is general - i.e. applicable to all instances of the problem, not just to the ones in the training set.

(This assumes, of course, that our space of problem instances is consistent - i.e. there aren't two different instances having the same description but different classifications.)
 3. One important issue that arises in conjunction with such a regimen is CONVERGENCE. How can we be sure that the training set will ultimately lead the program to deduce a rule that correctly classifies all instances of the problem?
 - a. Obviously, in the extreme, we could get a correct classification rule if we included every possible instance of the problem in our training set.
 - b. What we would like to know is whether we can achieve success by teaching the program using a randomly-selected subset of the set of all possible instances. In particular, can we say anything about how big this subset needs to be?
 4. The ultimate answer to this question is that we cannot guarantee success. What we can do, though, is establish some probabilistic measures of the likelihood of success. Here, we must consider two different issues:
 - a. The correctness of the classification rules we learn. In general, a set of classification rules can be quite useful even if they sometimes misclassify a particular observation.

b. The success of the learning process. A learning process can be quite useful if it leads the program to learn a good rule most of the time.

5. We say that a classification hypothesis that a program might learn is "approximately correct" to within some error epsilon if it classifies a randomly chosen instance incorrectly with probability \leq epsilon.

Example: We have previously seen that, for the arch-learning example we have used, the correct classification rule is:

$$On(b2, b1) \cap On(b2, b3) \cap \neg Adj(b1, b3)$$

Suppose that, in our world, only 3% of the instances had b1 and b3 adjacent. In this case, the following rule would be approximately correct, with epsilon 0.03:

$$On(b2, b1) \cap On(b2, b3)$$

6. We say that a learning process is probably approximately correct (PAC) for some epsilon (ϵ) if the learning process learns an approximately correct hypothesis (with error $\leq \epsilon$) most of the time. We can quantify "most of the the time" by some probability of failure to learn delta (δ).

Example: Suppose we wanted to learn a classification rule for arches that was approximately correct with error $\epsilon \leq 0.02$ - i.e. at least 98% of the time, it gives the correct classification. Suppose our particular learning regimen produced such a rule 95% of the time. Then we would say that the learning regimen is PAC with $\delta = 0.05$.

7. Poole develops an argument that shows that, for randomly chosen members of the training set, we can get a PAC result if we choose a random training set consisting of

$$(1/\epsilon) (\ln |H| + \ln 1/\delta) \quad \text{instances}$$

where |H| is the size of the hypothesis space - i.e. the number of possible different descriptions

Example: For the arch-learning problem, restricted to considering configurations of three blocks, and using the kind of descriptions we have been using, we have the following concepts, each of which may assume the values true or false.

$On(x, y)$ for each possible pair of blocks - 6 concepts

($On(b1, b2)$ is not the same as $On(b2, b1)$)

$Adj(x, y)$ for each possible pair of blocks - 3 concepts

($Adj(b1, b2)$ is the same as $Adj(b2, b1)$)

Hatched(X) for each possible block - 3 concepts

|H| is 2^{12} , since each term is either present or not present in a given description, and there are 12 possible terms.

(actually, the size of H is somewhat smaller, since $On(x, y)$ is impossible if $O(y, x)$)

To learn a rule that classifies potential arches 99% successfully ($\epsilon = .01$) at least 95% of the time ($\delta = .05$), we would have to use a training set of

$$(1/.01) * (\ln(2^{12}) + \ln(1/.05)) = 100 * (8.32 + 3) = 1132$$

randomly-chosen samples (or just 4 carefully chosen ones!)

IV. An Approach that Creates New Knowledge Structures

-- -- -----

- A. The final example of learning we will consider is one in which the program itself generates new knowledge categories. This is probably the most fascinating example we will consider.
- B. Here, the classic work is a program by Douglas Lenat called AM, which actually discovered a number of interesting mathematical concepts, such as the idea of prime numbers.
 - 1. READ DAVIS AND LENAT PP. 3-7; 10-12
 - 2. TRANSPARENCY: DAVIS AND LENAT PAGES 16, 22-23
 - 3. Note on the name - footnote on page 7.

V. Conclusion

- -----

- A. Learning remains a very difficult problem. Despite the successes that have been achieved in a few areas, it remains very much a research area.
- B. Perhaps the biggest problem is what Winston calls Martin's Law: "You can't learn anything unless you almost know it already". [This provides a rationale for incorporating a domain theory into a learner, as is done with explanation-based learning.]
- C. For our next major topic, we will look at an altogether different paradigm for learning using neural networks. One of the great attractions of neural networks is that they provide a framework that makes learning much more natural.