Unit-04: Learning

Artificial Intelligence

Dr. Mohammad Fayazur Rahaman
Associate Professor, `mfrahaman_ece@mgit.ac.in`

Dept. of Electronics and Communications Engineering,
Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad-75

Mar-Jun 2022

Unit-04: Learning [2]

Where are we ?

What is Learning ?

i. Machines cannot be called intelligent until they are able to learn to do new things and to adapt to new situations, rather than simply doing as they are told to do.

ii. The ability to adapt to new surroundings and to solve new problems is an important characteristic of intelligent entities.

iii. Simon has proposed that learning denotes "`changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time`".

iv. Learning covers a wide range of phenomena. At one end of the spectrum is skill refinement. People get better at many tasks simply by practicing.

v. At the other end of the spectrum lies knowledge acquisition. As we have seen, many AI programs draw heavily on knowledge as their source of power. Knowledge is generally acquired through experience.

vi. Knowledge acquisition itself includes many different activities. Simple storing of computed information, or **rote learning**, is the most basic learning activity.

vii. Many computer programs, e.g., database systems, can be said to "learn" in this sense, although most people would not call such simple storage learning. However, many AI programs are able to improve their performance substantially through rote-learning techniques,

viii. Another way we learn is through **taking advice** from others. Advice taking is similar to rote learning, but high-level advice may not be in a form simple enough for a program to use directly in problem solving. The advice may need to be first operationalized.

ix. People also learn through their own **problem-solving experience**. After solving a complex problem, we remember the structure of the problem and the methods we used to solve it. The next time we see the problem, we can solve it more efficiently.

x. In contrast to advice taking, learning from problem-solving experience does not usually involve gathering new knowledge that was previously unavailable to the learning program.

xi. Another form of learning that does involve stimuli from the outside is **learning from examples**. We often learn to classify things in the world without being given explicit rules. For example, adults can differentiate between cats and dogs, but small children often cannot.

Where are we ?

Rote Learning

i. When a computer stores a piece of data, it is performing a rudimentary form of learning. Caching has been used in AI programs to produce some surprising performance improvements. Such caching is known as rote learning.

ii. One of the earliest game-playing programs, checkers program learned to play checkers well enough to beat its creator. It exploited two kinds of learning: rote learning, and parameter (or coefficient) adjustment

iii. This program used the minimax search procedure to explore checkers game trees.
- As is the case with all such programs, time constraints permitted it to search only a few levels in the tree.
- When it could search no deeper, it applied its static evaluation function to the board position and used that score to continue its search of the game tree.
- When it finished searching the tree and propagating the values backward, it had a score for the position represented by the root of the tree.
- It could then choose the best move and make it. But it also recorded the board position at the root of the tree and the backed up score that had just been computed for it.

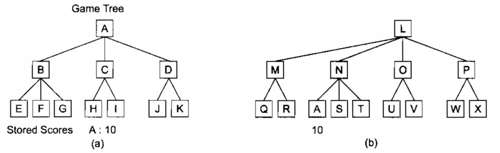

Fig.17.1 *Storing Backed-Up Values*

iv. Rote learning of this sort is very simple. It does not appear to involve any sophisticated problem-solving capabilities.

v. But even it shows the need for some capabilities that will become increasingly important in more complex learning systems.

vi. These capabilities include:

    I. **Organized Storage of Information**- In order for it to be faster to use a stored value than it would be to recompute it, there must be a way to access the appropriate stored value quickly.

    II. **Generalization**-The number of distinct objects that might potentially be stored can be very large. To keep the number of stored objects down to a manageable level, some kind of generalization is necessary.

    - The number of distinct objects that could be stored was equal to the number of different board positions that can arise in a game.
    - As the complexity of the learning process increases, so too does the need for generalization.

Where are we ?

Learning by Taking Advice

i. A computer can do very little without a program for it to run. When a programmer writes a series of instructions into a computer, a rudimentary kind of learning is taking place:
   - The programmer is a sort of teacher, and the computer is a sort of student. After being programmed, the computer is now able to do something it previously could not.
   - Executing the program may not be such a simple matter, however. Suppose the program is written in a high-level language like LISP. Some interpreter or compiler must intervene to change the teacher's instructions into code that the machine can execute directly.

ii. People process advice in an analogous way. In chess, the advice "fight for control of the center of the board" is useless unless the player can translate the advice into concrete moves and plans. A computer program might make use of the advice by adjusting its static evaluation function to include a factor based on the number of center squares attacked by its own pieces.

iii. Mostow describes a program called FOO, which accepts advice for playing hearts, a card game. A human user first translates the advice from English into a representation that FOO can understand.

iv. For example, "Avoid taking points" becomes: (avoid (take-points me) (trick))

v. FOO must operationalize this advice by turning it into an expression that contains concepts and actions FOO can use when playing the game of hearts.

vi. In other words, when playing a card that is the same suit as the card that was played first, if the trick possibly contains points, then play a low card.

vii. At last, FOO has translated the rather vague advice "avoid taking points" into a specific, usable heuristic.

viii. A human can watch FOO play, detect new mistakes, and correct them through yet more advice, such as "play high cards when it is safe to do so." The ability to operationalize knowledge is critical for systems that learn from a teacher's advice.

Where are we ?

Learning in Problem Solving

In the last section, we saw how a problem solver could improve its performance by taking advice from a teacher. Can a program get better without the aid of a teacher? It can, by generalizing from its own experiences.

Learning by Parameter Adjustment

i. Many programs rely on an evaluation procedure that combines information from several sources into a single summary statistic.
ii. Game-playing programs do this in their static evaluation functions, in which a variety of factors, such as piece advantage and mobility are combined into a single score reflecting the desirability of a particular board position.
iii. Pattern classification programs often combine several features to determine the correct category into which a given stimulus should be placed.
iv. In designing such programs, it is often difficult to know a priori how much weight should be attached to each feature being used.
v. One way of finding the correct weights is to begin with some estimate of the correct settings and then to let the program modify the settings on the basis of its experience.
vi. Features that appear to be good predictors of overall success will have their weights increased, while those that do not will have their weights decreased
vii. Checkers program exploited this kind of learing in addition to the rote learning described above, and it provides a good example of its use.
viii. As its static evaluation function, the program used a polynomial of the form

$$c_1t_1 + c_2t_2 + \cdots + c_{16}t_{16}$$

ix. The t terms are the values of the sixteen features that contribute to the evaluation
x. The c terms are the coefficients (weights) that are attached to each of these values.
   - As learning progresses, the c values will change
xi. The most important question in the design of a learning program based on parameter adjustment is "When should the value of a coefficient be increased and when should it be decreased?"
xii. The second question to be answered is then "By how much should the value be changed?"
xiii. The simple answer to the first question is that the coefficients of terms that predicted the final outcome accurately should be increased, while the coefficients of poor predictors should be decreased.
xiv. In some domains, this is easy to do.
   - If a pattern classification program uses its evaluation function to classify an input and it gets the right answer, then all the terms that predicted that answer should have their weights increased.
   - But in game-playing programs, the problem is more difficult. The program does not get any concrete feedback from individual moves.
      - It does not find out for sure until the end of the game whether it has won.
      - But many moves have contributed to that final outcome.
      - Even if the program wins, it may have made some bad moves along the way.
      - The problem of appropriately assigning responsibility to each of the steps that led to a single outcome is known as the credit assignment problem.
xv. Samuel's program exploits one technique, albeit imperfect, for solving this problem.
   - Assume that the initial values chosen for the coefficients are good enough that the total evaluation function produces values that are fairly reasonable measures of the correct score even if they are not as accurate as we hope to get them.
   - Then this evaluation function can be used to provide feedback to itself. Move sequences that lead to positions with higher values can be considered good (and the terms in the evaluation function that suggested them can be reinforced).

xvi. Because of the limitations of this approach, however, Samuel's program did two other things, one of which pròvided an additional test that progress was being made and the other of which generated additional nudges to keep the process out of a rut:

   I. When the program was in learning mode, it played against another copy of itself.
      - Only one of the copies altered its scoring function during the game; the other remained fixed.
      - At the end of the game, if the copy with the modified function won, then the modified function was accepted. Otherwise, the old one was retained.
      - If, however, this happened very many times, then some drastic change was made to the function in an attempt to get the process going in a more profitable direction.

   II. Periodically, one term in the scoring function was eliminated and replaced by another.

xvii. This process of learning by successive modifications to the weights of terms in a scoring function has many limitations, mostly arising out of its lack of exploitation of any knowledge about the structure of the problem with which it is dealing and the logical relationships among the problem's components.

xviii. In addition, because the learning Procedure is a variety of hill climbing, it suffers from the same difficulties as do other hill climbing programs.

xix. Parameter adjustment is certainly not a solution to the overall learning problem. But it is often a useful technique, either in situations where very little additional knowledge is available or in programs in which it is combined with more knowledge-intensive methods

Where are we ?

Learning from Examples

i. Classification is the process of assigning, to a particular input, the name of a class to which it belongs.

ii. Classification is an important component of many problem-solving tasks. In its simplest form, it is presented as a straightforward recognition task.

iii. Consider a problem-solving system that contains the following production rule:
   - `If: the current goal is to get from place A to place B, and there is a WALL separating the two places`
   - `then: look for a DOORWAY in the WALL and go through it.`

iv. To use this rule successfully, the system's matching routine must be able to identify an object as a wall. Without this, the rule can never be invoked.

v. Then, to apply the rule, the system must be able to recognize a doorway.

vi. Before classification can be done, the classes it will use must be defined. This can be done in a variety of ways, including:

   I. Isolate a set of features that are relevant to the task domain.

- Define each class by a weighted sum of values of these features. Each class is then defined by a scoring function that looks very similar to the scoring functions often used in other situations, such as game playing. Such a function has the form:

$$c_1t_1 + c_2t_2 + c_3t_3 + \cdots$$

- Each t corresponds to a value of a relevant parameter, and each c represents the weight to be attached to the corresponding . Negative weights can be used to indicate features whose presence usually constitutes negative evidence for a given class.
- For example, if the task is weather prediction, the parameters can be such measurements as rainfall and location of cold fronts.
- Different functions can be written to combine these parameters to predict sunny, cloudy, rainy, or snowy weather.

II. Isolate a set of features that are relevant to the task domain. Define each class as a structure composed of those features.
   - For example, if the task is to identify animals, the body of each type of animal can be stored as a structure, with various features representing such things as color, length of neck, and feathers.

vii. There are advantages and disadvantages to each of these general approaches.

viii. The statistical approach taken by the first scheme presented here is often more efficient than the structural approach taken by the second.

ix. But the second is more flexible and more extensible.

x. Regardless of the way that classes are to be described, it is often difficult to construct, by hand, good class definitions. This is particularly true in domains that are not well understood or that change rapidly.

xi. Thus the idea of producing a classification program that can evolve its own class definitions is appealing. This task of constructing class definitions is called concept learning, or induction.

xii. The techniques used for this task must, of course, depend on the way that classes (concepts) are described.

Where are we ?

Winston's Learning Program

i. Winston describes an early structural concept learning program. This program operated in a simple blocks world domain.

ii. Its goal was to construct representations of the definitions of concepts in the blocks domain.
- For example, it learned the concepts House, Tent, and Arch shown in Figure
- The figure also shows an example of a near miss for each concept.
- A near miss is an object that is not an instance of the concept in question but that is very similar to such instances.

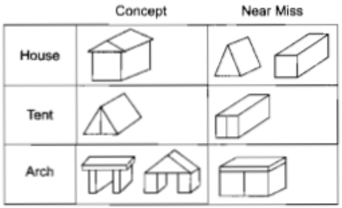

Fig. 17.2 *Some Blocks World Concepts*

iii. The program started with a line drawing of a blocks world structure.

iv. This structural description was then provided as input to the learning program. An example of such a structural description for the House is shown in Figure below.

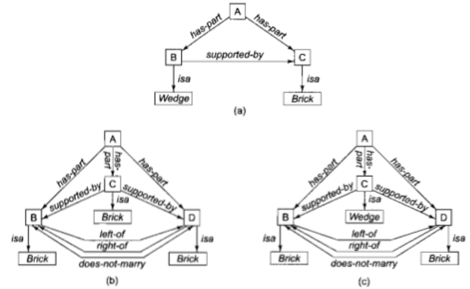

Fig. 17.3 *Structural Descriptions*

v. Node A represents the entire structure, which is composed of two parts: node B, a Wedge, and node C, a Brick.

vi. Figures (b) and (c) show descriptions of the two Arch structures

vii. These descriptions are identical except for the types of the objects on the top; one is a Brick while the other is a Wedge.

viii. Notice that the two supporting objects are related not only by left-of and right-of links, but also by a does-not-marry link, which says that the two objects do not marry.

ix. Two objects marry if they have faces that touch and they have a common edge.

x. The marry relation is critical in the definition of an Arch. It is the difference between the first arch structure and the near miss arch structure

xi. It is the difference between the first arch structure and the near miss arch structure shown

xii. The basic approach that Winston's program took to the problem of concept formation can be described as follows:

    I. Begin with a structural description of one known instance of the concept. Call that description the concept definition.
    II. Examine descriptions of other known instances of the concept. Generalize the definition to include them.
    III. Examine descriptions of near misses of the concept. Restrict the definition to exclude these.

Where are we ?

Decision Trees

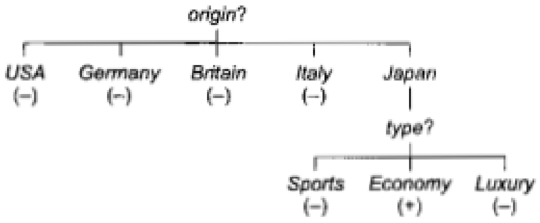

Fig. 17.13 *A Decision Tree*

i. A third approach to concept learning is the induction of decision trees
ii. To classify a particular input, we start at the top of the tree and answer questions until we reach a leaf, where the classification is stored.
iii. Figure represents the familiar concept "Japanese economy car."
iv. ID3 is a program that builds decision trees automatically, given positive and negative instances of a concept.
v. ID3 uses an iterative method to build up decision trees, preferring simple trees over complex ones, on the theory that simple trees are more accurate classifiers of future inputs.
vi. It begins by choosing a random subset of the training, examples. This subset is called the window.
vii. The algorithm builds a decision tree that correctly classifies all examples in the window.
viii. The tree is then tested on the training examples outside the window.
ix. If all the examples are classified correctly, the algorithm halts. Otherwise, it adds a number of training examples to the window and the process repeats.
x. Empirical evidence indicates that the iterative strategy is more efficient than considering the whole training set at once.

Text Books

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*.
Third Edition, Prentice-Hall, 2009.

[2] S. N. Elaine Rich, Kevin Knight, *Artificial Intelligence*.
Third Edition, The McGraw Hill Publications, 2009.

[3] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*.
Sixth Edition, Pearson Education, 2009.

Thank you