Unit 12 Learning and Artificial Intelligence

Structure:

12.1 Introduction

In the previous unit, you have learnt the design of an expert system with the help of Mycin and Dendral. Learning is distinguished into a number of different forms. The simplest is learning by trial-and-error. For example, a simple program for solving mate-in-one chess problems might try out moves at random until one is found that achieves mate. The program remembers the successful move and next time the computer is given the same problem it is able to produce the answer immediately. The simple memorizing of individual items – solutions to problems, words of vocabulary, etc. – is known as rote learning.

In this unit, you will be introduced to the concept of learning and its models. You will also be introduced to the concept of theory driven discovery. One of the most often heard criticisms of AI is that 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. There can be questions that the ability to adapt to new surroundings and to solve new problems is an important characteristic of intelligent entities. Rather than asking in advance whether it is possible for computers to "learn," it is much more enlightening to try to describe exactly what activities we mean when we say "learning" and what mechanisms could be used to enable us to perform those activities.

Objectives:

After studying this unit, you should be able to

- define learning
- list the different approaches by which we can learn
- explain factors affecting learning performance
- describe Rote Learning, and Learning by Taking Advice.
- explain Learning by Problem Solving
- explain Inductive and Explanation Based Learning
- discuss the concepts of Discovery and Analogy with respect to AI.

12.2 An Introduction to Learning

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. The more we ride a bicycle or play tennis, the better we get. At the other end of the spectrum lies *knowledge acquisition.*

AI programs draw heavily on knowledge as their source of power. Knowledge is generally acquired through experience, and such acquisition is the focus of this unit along with Robotics.

Knowledge acquisition itself includes many different activities. Simple storing of computed information, or *rote learning,* is the most basic learning activity. 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. 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. Moreover, we can generalize from our experience to solve related problems more easily. In contrast to advice taking, learning from problem-solving experiences does not usually involve gathering new knowledge that was previously unavailable to the learning program. That is, the program remembers its experiences and generalizes from them, but does not add to the transitive closure of its knowledge, in the sense that an advice-taking program would, i.e., by receiving stimuli from the outside world. In large problem spaces, however, efficiency gains are critical. Practically speaking, learning can mean the difference between solving a problem rapidly and not solving it at all. In addition, programs that learn through problem-solving experience may be able to come up with qualitatively better solutions in the future.

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. Somewhat along the line, we induce a method for telling cats from dogs based on seeing numerous examples of each. Learning from examples usually involves a teacher who helps us classify things by correcting us when we are wrong. Sometimes, however, a program can discover things without the aid of a teacher.

AI researchers have proposed many mechanisms for doing the kinds of learning described above. In this unit, we discuss several of them.

Learning is an important area in AI, perhaps more so than planning.

- Problems are hard – harder than planning.
- Recognized Solutions are not as common as planning.
- A goal of AI is to enable computers that can be taught rather than programmed.

Learning is an area of AI that focuses on processes of self-improvement.

Information processes that improve their performance or enlarge their knowledge bases are said to *learn*.

Why is it hard?

- Intelligence implies that an organism or machine must be able to adapt to new situations.
- It must be able to learn to do new things.
- This requires knowledge acquisition, inference, updating/refinement of knowledge base, acquisition of heuristics, applying faster searches, *etc.*

12.2.1 How can we learn?

Many approaches have been taken to attempt to provide a machine with learning capabilities. This is because learning tasks cover a wide range of phenomena.

Listed below are a few examples of how one may learn. We will look at these in detail shortly.

Skill refinement
– one can learn by practicing, *e.g playing the piano*.

Knowledge acquisition
– one can learn by experience and by storing the experience in a knowledge base. One basic example of this type is rote learning.

Taking advice
– Similar to rote learning although the knowledge that is input may need to be transformed (or *operationalised*) in order to be used effectively.

Problem Solving
– if we solve a problem one may learn from this experience. The next time we see a similar problem we can solve it more efficiently. This does not usually involve gathering new knowledge but may involve reorganisation of data or remembering how to achieve to solution.

Induction
– One can learn from *examples*. Humans often classify things in the world without knowing explicit rules. Usually involves a teacher or trainer to aid the classification.

Discovery

– Here one learns knowledge without the aid of a teacher.

Analogy

– If a system can recognise similarities in information already stored then it may be able to transfer some knowledge to improve to solution of the task in hand.

12.2.2 General Learning Model

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 can 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 12.1 where the environment has been included as part of the overall learner system.

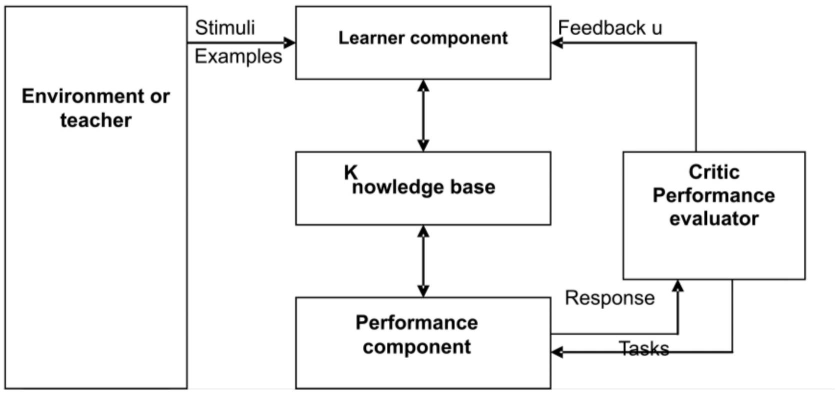

Figure 12.1: General Learning Model

The environment may be regarded as either a form of nature which produces random stimuli or as a more 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 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 the 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 can 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.

12.2.3 Factors affecting Learning Performance

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 algorithm used. It is shown in Figure 12.2.

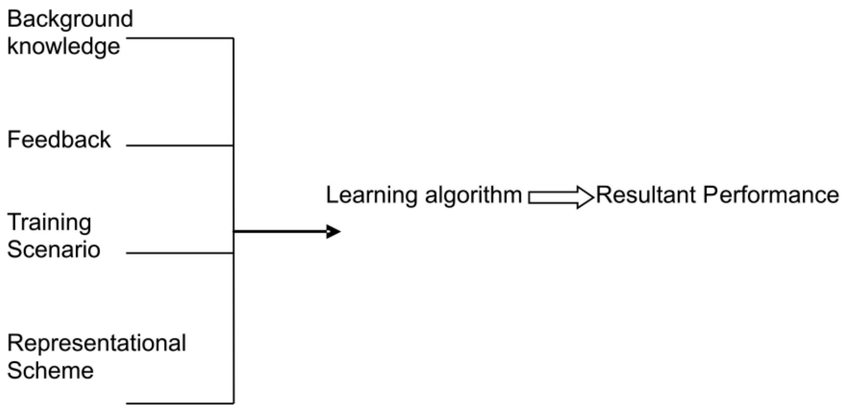

Figure 12.2: Factors affecting learning performance

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 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. 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.

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 algorithm that extract much of the useful information from training examples and take advantage of any background knowledge outperform those that do not.

Self Assessment Questions

1. _________ cannot be called intelligent until they are able to learn to do new things and to adapt to new situations.
2. Which is the form of learning that does involve stimuli from the outside? ____
3. The actual form of environment used will depend on the particular __________ paradigm.
4. Many forms of learning cannot be characterized as a search through a space of possible hypotheses or solutions. (State true or false)

12.3 Rote Learning

Rote Learning is basically *memorisation*.

- Saving knowledge so it can be used again.
- Retrieval is the only problem.
- No repeated computation, inference or query is necessary.

A simple example of rote learning is *caching*. Its features are:

- Store computed values (or large piece of data)
- Recall this information when required by computation.
- Significant time savings can be achieved.
- Many AI programs (as well as more general ones) have used caching very effectively.

Memorization is a key necessity for learning:

- It is a basic necessity for any intelligent program – is it a separate learning process?
- Memorization can be a complex subject – how best to store knowledge?

Samuel's Checkers program employed rote learning (it also used parameter adjustment).

- A minimax search was used to explore the game tree.
- Time constraints do not permit complete searches.
- It *records* board positions and scores at search ends.
- Now if the same board position arises later in the game the stored value can be recalled and the end effect is that more deeper searches have occurred.

Rote learning is basically a simple process. However it does illustrate some issues that are relevant to more complex learning issues.

Organisation*:*

-- access of the stored value must be faster than it would be to recompute it. Methods such as hashing, indexing and sorting can be employed to enable this.

E.g Samuel’s program indexed board positions by noting the number of pieces.

Generalisation*:*

– The number of potentially stored objects can be very large. We may need to generalise some information to make the problem manageable.

E.g Samuel’s program stored game positions only for white to move. Also rotations along diagonals are combined.

Stability of the environment:

– Rote learning is not very effective in a rapidly changing environment. If the environment does change then we must detect and record exactly what has changed – *the frame problem*.

Store v Compute

Rote Learning must not decrease the efficiency of the system.

We must be able to decide whether it is worth storing the value in the first place.

Consider the case of multiplication – it is quicker to recompute the product of two numbers rather than store a large multiplication table.

How can we decide?

Cost-benefit analysis
– Decide when the information is first available whether it should be stored. An analysis could weigh up amount of storage required, cost of computation, likelihood of recall.

Selective forgetting
– Here we allow the information to be stored initially and decide later if we retain it. Clearly the frequency of reuse is a good measure. We could tag an object with its *time of last use*. If the cache memory is full and we wish to add a new item we remove the least recently used object. Variations could include some form of cost-benefit analysis to decide if the object should be removed.

12.4 Leaning by Taking Advice

The idea of advice taking in AI based learning was proposed as early as 1958 (McCarthy). However very few attempts were made in creating such systems until the late 1970s. Expert systems are providing a major impetus in this area.

There are two basic approaches to advice taking:

- Take high level, abstract advice and convert it into rules that can guide performance elements of the system. *Automate all aspects of advice taking.*
- *Develop sophisticated tools* such as knowledge base editors and debugging. These are used to aid an expert to translate his expertise into detailed rules. Here the expert is an *integral* part of the learning system. Such tools are important in *expert systems* area of AI.

12.4.1 Automated advice taking

The following steps summarise this method:

Request
– This can be a simple question asking about general advice or more complicated by identifying shortcomings in the knowledge base and asking for a remedy.

Interpret
– Translate the advice into an *internal representation*.

Operationalise
– Translated advice may still not be usable so this stage seeks to provide a representation that can be used by the performance element.

Integrate
– When knowledge is added to the knowledge base care must be taken so that bad side-effects are avoided.
E.g. Introduction of redundancy and contradictions.

Evaluate
– The system must assess the new knowledge for errors, contradictions *etc.*
The steps can be iterated.

12.4.2 Knowledge base maintenance

Instead of automating the five steps above, many researchers have instead assembled tools that aid the development and maintenance of the knowledge base.

Many have concentrated on:
- Providing intelligent editors and flexible representation languages for integrating new knowledge.
- Providing debugging tools for evaluating, finding contradictions and redundancy in the existing knowledge base.

EMYCIN is an example of such a system.

Self Assessment Questions
5. Rote Learning is basically ________________.
6. ______________ we allow the information to be stored initially and decide later if we retain it.
7. The idea of advice taking in AI based learning was proposed as early as ____________

12.5 Learning by Problem Solving

There are three basic methods in which a system can learn from its own experiences.
- Learning by Parameter Adjustment

- Learning by Macro Operators
- Learning by Chunking

12.5.1 Learning by parameter adjustment

Many programs rely on an evaluation procedure to summarise the state of search *etc.* Game playing programs provide many examples of this.

However, many programs have a static evaluation function.

In learning, a slight modification of the formulation of the evaluation of the problem is required.

Here the problem has an evaluation function that is represented as a polynomial of the form such as:

$$c_1t_1 + c_2t_2 + c_3t_3 + \dots$$

The *t* terms are values of features and the *c* terms are weights.
In designing programs it is often difficult to decide on the exact value to give each weight initially.

So the basic idea of *parameter adjustment* is to:

- Start with some estimate of the correct weight settings.
- Modify the weight in the program on the basis of accumulated experiences.
- Features that appear to be good predictors will have their weights increased and bad ones will be decreased.

Samuel's Checkers programs employed 16 such features at any one time chosen from a pool of 38.

12.5.2 Learning by macro operators

The basic idea here is similar to Rote Learning:

Avoid expensive recomputation

Macro-operators can be used to group a whole series of actions into one.

For example: Making dinner can be described as lay the table, cook dinner, serve dinner. We could treat laying the table as on action even though it involves a sequence of actions.

The STRIPS problem-solving employed macro-operators in it's learning phase.

Consider a blocks world example in which ON(C,B) and ON(A,TABLE) are true.

STRIPS can achieve ON(A,B) in four steps:

UNSTACK(C,B), PUTDOWN(C), PICKUP(A), STACK(A,B)

STRIPS now builds a macro-operator MACROP with preconditions ON(C,B), ON(A,TABLE), post conditions ON(A,B), ON(C,TABLE) and the four steps as its body.

MACROP can now be used in future operation.

But it is not very general. The above can be easily generalised with variables used in place of the blocks. However generalisation is not always that easy.

12.5.3 Learning by chunking

Chunking involves similar ideas to Macro Operators and originates from psychological ideas on memory and problem solving.

The computational basis is in production systems (studied earlier).

SOAR is a system that uses production rules to represent its knowledge. It also employs chunking to learn from experience.

Basic outline of SOAR's method

- SOAR solves problems, it fires productions, these are stored in *long term memory*.
- Some firings turn out to be more useful than others.
- When SOAR detects are useful sequence of firings, it creates *chunks*.
- A *chunk* is essentially a large production that does the work of an entire sequence of smaller ones.
- Chunks may be generalised before storing.

12.6 Inductive and Explanation Based Learning

In this section you will be introduced to the learning methods namely inductive learning and explanation based learning.

12.6.1 Inductive learning

This involves the process of *learning by example* -- where a system tries to induce a general rule from a set of observed instances.

This involves classification -- assigning, to a particular input, the name of a class to which it belongs. Classification is important to many problem solving tasks.

A learning system has to be capable of evolving its own class descriptions:

- Initial class definitions may not be adequate.
- The world may not be well understood or rapidly changing.

The task of constructing class definitions is called *induction* or *concept learning.*

12.6.2 Explanation Based Learning (EBL)

Humans appear to learn quite a lot from one example.

Basic idea: Use results from one examples problem solving effort next time around.

An EBL accepts 4 kinds of input:

A training example
– what the learning *sees* in the world.

A goal concept
– a high level description of what the program is supposed to learn.

A operational criterion
– a description of which concepts are usable.

A domain theory
– a set of rules that describe relationships between objects and actions in a domain.

From this EBL computes a generalisation of the training example that is sufficient not only to describe the goal concept but also satisfies the operational criterion.

This has two steps:

Explanation
– the domain theory is used to prune out all unimportant aspects of the training example with respect to the goal concept.

Generalisation
– the explanation is generalised as far possible while still describing the goal concept.

Self Assessment Questions

8. In designing programs it is often difficult to decide on the exact value to give each weight initially. (Say Yes or No)
9. The task of constructing ________ definitions is called induction or concept learning.

12.7 Discovery

Discovery is a restricted form of learning in which one entity acquires knowledge without the help of a teacher.

These are of two types:

- Theory Driven Discovery - AM (1976)
- Data Driven Discovery -- BACON (1981)

12.7.1 Theory driven discovery – AM (1976)

AM is a program that discovers concepts in elementary mathematics and set theory.

AM has 2 inputs:

- A description of some concepts of set theory (in LISP form). *E.g.* set union, intersection, the empty set.
- Information on how to perform mathematics. *E.g.* functions.

Given the above information AM *discovered*:

Integers
– it is possible to count the elements of this set and this is an the image of this counting function -- the integers -- interesting set in its own right.

Addition
– The union of two disjoint sets and their counting function.

Multiplication
– Having discovered addition and multiplication as laborious set-theoretic operations more effective descriptions were supplied by hand.

Prime numbers

– factorisation of numbers and numbers with only one factor were discovered.

Golbach's conjecture

– Even numbers can be written as the sum of 2 primes. *E.g.* 28 = 17 + 11.

Maximally divisible numbers

– numbers with as many factors as possible. A number *k* is maximally divisible if *k* has more factors than any integer less than *k*. *E.g.* 12 has six divisors 1,2,3,4,6,12.

How does AM work?

AM employs many general-purpose AI techniques:

- A frame based representation of mathematical concepts.
  - AM can create new concepts (slots) and fill in their values.
- Heuristic search employed
  - 250 heuristics represent *hints* about activities that might lead to interesting discoveries.
  - How to employ functions, create new concepts, generalisation *etc.*
- Hypothesis and test based search.
- Agenda control of discovery process.

12.7.2 Data driven discovery – BACON (1981)

Many discoveries are made from observing data obtained from the world and making sense of it – *E.g.* Astrophysics – discovery of planets, Quantum mechanics – discovery of sub-atomic particles.

BACON system outline:

- Starts with a set of variables for a problem.
  - *E.g.* BACON was able to derive the *ideal gas law*. It started with four variables p - gas pressure, V – gas volume, n – molar mass of gas, T – gas temperature. Recall $pV/nT = k$ where k is a constant.
- Values from experimental data from the problem which are inputted.
- BACON holds some constant and attempts to notice trends in the data.
- Inferences made.

BACON has also been applied to Kepler's 3rd law, Ohm's law, conservation of momentum and Joule's law.

12.8 Analogy

Analogy involves a complicated mapping between what might appear to be two dissimilar concepts.

Bill is built like a large outdoor brick lavatory.

He was like putty in her hands

Humans quickly recognise the abstractions involved and understand the meaning.

There are two methods of analogical problem methods studied in AI:

- Transformational Analogy
- Derivational Analogy

12.8.1 Transformational analogy

Look for a similar solution and *copy* it to the new situation making suitable substitutions where appropriate.

E.g. Geometry.

If you know about lengths of line segments and a proof that certain lines are equal (See figure. 12.3) then we can make similar assertions about angles.

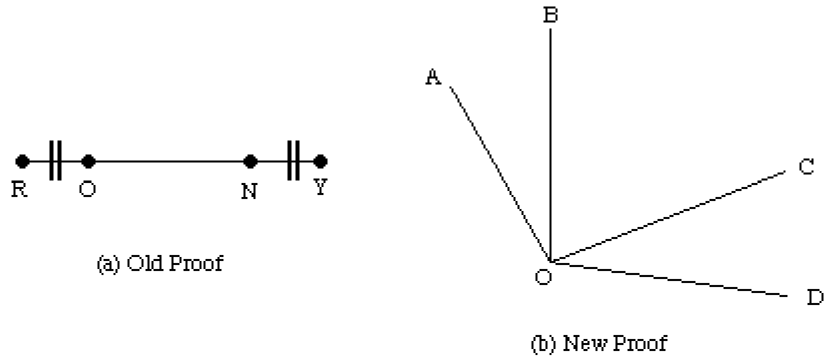

Figure 12.3: Transformational Analogy Example

- We know that lines *RO* = *NY* and angles *AOB* = *COD*
- We have seen that *RO* + *ON* = *ON* + *NY* – additive rule.
- So we can say that angles *AOB* + *BOC* = *BOC* + *COD*
- So by a transitive rule line *RN* = *OY*
- So similarly angle *AOC* = *BOD*

Carbonell (1983) describes a *T-space* method to transform old solutions into new ones.

- Whole solutions are viewed as states in a problem space – the *T-space*.
- *T-operators* prescribe methods of transforming existing solution states into new ones.
- Reasoning by analogy becomes a search in T-space – means-end analysis.

12.8.2 Derivational analogy

Transformational analogy does not look at how the problem was solved – it only looks at the final solution.

The *history* of the problem solution – the steps involved – are often relevant.

Carbonell (1986) showed that derivational analogy is a necessary component in the transfer of skills in complex domains:

- In translating Pascal code to LISP – line by line translation is no use. You will have to *reuse* the major structural and control decisions.
- One way to do this is to *replay* a previous derivation and modify it when necessary.
- If initial steps and assumptions are still valid copy them across.
- Otherwise alternatives need to found – best first search fashion.

Self Assessment Questions

10. ________ is a program that discovers concepts in elementary mathematics and set theory.
11. ____________ involves a complicated mapping between what might appear to be two dissimilar concepts.
12. Who showed that derivational analogy is a necessary component in the transfer of skills in complex domains?

12.9 Summary

In this unit, you have learnt the various aspects of learning and its models. You have also learnt the concepts like discovery and analogy. Knowledge is generally acquired through experience. Learning is an area of AI that focuses on processes of self-improvement Many AI programs are able to improve their performance substantially through rote-learning techniques. Learning from examples is another form of learning that does involve stimuli from the outside. Learning requires that new knowledge structures can be created from some form of input stimulus. There are several important factors which influence a system's ability to learn in addition to the form of representation used. In Selective forgetting we allow the information to be stored initially and decide later if we retain it. In learning, a slight modification of the formulation of the evaluation of the problem is required. Discovery is a restricted form of learning in which one entity acquires knowledge without the help of a teacher. Analogy involves a complicated mapping between what might appear to be two dissimilar concepts.

12.10 Terminal Questions

1. What is learning? Explain briefly.
2. What are the different approaches by which we can learn?
3. What are the factors that affect learning performance?
4. Explain Inductive and Explanation Based Learning
5. What do you mean by Discovery? Explain types of Discovery
6. Write a note on Analogy

12.11 Answers

Self Assessment Questions

1. Machines
2. learning from examples.
3. Learning
4. False
5. Memorisation
6. Selective forgetting
7. 1958
8. Yes
9. Class
10. AM
11. Analogy
12. Carbonell

Terminal Questions

1. Learning is an area of AI that focuses on processes of self-improvement (Refer section 12.2)
2. There are different approaches by which we can learn. For example skill refinement. (Refer sub-section 12.2.1)
3. The factors which affect learning performance include the types of training provided etc., (Refer sub-section 12.2.3)
4. Inductive learning: This involves the process *of* learning by example where a system tries to induce a general rule from a set of observed instances. (Refer sub-sections 12.6.1 and 12.6.2)
5. Discovery is a restricted form of learning in which one entity acquires knowledge without the help of a teacher. (Refer section 12.7)
6. Analogy involves a complicated mapping between what might appear to be two dissimilar concepts. (Refer section 12.8)