UNIT IV PLANNING AND MACHINE LEARNING

Basic plan generation systems - Strips -Advanced plan generation systems – K strips -Strategic explanations -Why, Why not and how explanations. Learning - Machine learning, adaptive Learning.

PLANNING

Planning refers to the process of computing several steps of a problem solving procedure before executing any of them.

STRIPS

- Its name is derived from STanford Research Institute Problem Solver.
- An automated planner.
- Planning system for a robotics project: SHAKEY.
- Developed by Richard Fikes and Nils Nilsson in 1971 at SRI International.
- Classical Planning System.
- Knowledge Representation: First Order Logic.
- Algorithm: Forward chaining on rules.

STRIPS-Like Planning Formulation

- A finite, nonempty set of *instances*.
- A finite, nonempty set of *predicates*, which are binary-valued (partial) functions of one of more instances. Each application of a predicate to a specific set of instances is called a *positive literal*. A logically negated positive literal is called a *negative literal*.
- A finite, nonempty set of *operators*, each of which has:

  1) *preconditions*, which are positive or negative literals that must hold for the operator to apply, and

  2) *effects*, which are positive or negative literals that are the result of applying the operator.
- An *initial set* which is expressed as a set of *positive literals*. Negative literals are implied. For any positive literal that does not appear in , its corresponding negative literal is assumed to hold initially.
- A *goal set* which is expressed as a set of both *positive* and *negative literals*.

STRIPS Planner

- STRIPS maintains **two additional data structures**:
  - **State List** - all currently true predicates.
  - **Goal Stack** - a push down stack of goals to be solved, with current goal on top of stack. *f*
- If the current goal is not satisfied by present state,
  - Find goal in the add list of an operator, and push operator and preconditions list on stack. (=Sub goals).
- When a current goal is satisfied, OP it from stack. *f*
- When an operator is on top of the stack,
  - record the application of that operator – update the plan sequence and
  - use the operator's add and delete lists to update the current state.

Reasoning Loop

- If the top item on the goal stack is:
  - empty (the goal stack is empty), return the **actions** executed
    - they form the plan to achieve the goal
  - a **goal**, and it is **satisfied** in the current state, remove it from the stack (no replacement necessary)
  - a complex goal, break it into sub goals, placing all sub goals on the goal stack (the original goal is pushed down into the goal stack)
  - a predicate, find an action that will make it true, then place that action (with variables bound appropriately) and its preconditions on the goal stack (preconditions first)

- an action and its preconditions are satisfied, perform the action, updating the world state using the delete and add lists of the action (if the pre-conditions are not satisfied, add them to the goal list without removing the action).
- Add this action to the partial plan.

STRIPS Limitation

- Expressive power of description language for the operators:
  - Operator changes exactly, what is specified in its add list and delete list.
- Problems with the strategy: Sussman anomaly

EXAMPLE-BLOCKS WORLD (BW) PLANNING

- The world consists of:
  - **A flat surface** such as a table top
  - An adequate **set of identical blocks** which are **identified by letters**.
  - The blocks can be **stacked** one on one to form towers of apparently unlimited height.

Why Use the Blocks World as an Example?

- Sufficiently simple and well behaved.
- Easily understood.
- Good sample environment to study planning.
  - Problems can be broken into nearly distinct sub problems.
  - We can show how partial solutions need to be combined to form a realistic complete solution.

The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations. The robot can hold one block at a time and only one block can be moved at a time. Any number of blocks can be on the table. The actions it can perform include

- *stack(X,Y)*: put block X on block Y. The arm must already be holding X and the surface of Y must be clear.
- *unstack(X,Y)*: remove block X from block Y. The arm must be empty and block X must have no blocks on top of it.
- *pickup(X)*: pickup block X from the table. The arm must be empty and there must be nothing on top of X.
- *putdown(X)*: put block X on the table. The arm must have been holding block X. *f*

The **stack action** occurs when the robot arm places the object it is holding [X] on top of another object [Y].

- **FORM:** STACK(X,Y) □
- **PRE:** CLEAR(Y) ∧ HOLDING(X) □
- **ADD:** ARMEMPTY() ∧ ON(X,Y) ∧ CLEAR(X) □
- **DEL:** CLEAR(Y) ∧ HOLDING(X) □
- **CONSTRAINTS:** (X ≠ Y), X ≠ TABLE, Y ≠ TABLE

The **unstack action** occurs when the robot arm picks up an object X from on top of another object Y.

- **FORM:** UNSTACK(X,Y) □
- **PRE:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) ∧ CLEAR(Y) □
- **DEL:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **CONSTRAINTS:** X ≠ Y, X ≠ TABLE, Y ≠ TABLE

The **pickup action** occurs when the arm picks up an object (block) X from the table. □

- **FORM:** PICKUP(X) □
- **PRE:** ONTABLE(X) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) □
- **CONSTRAINTS:** X ≠ TABLE

The **putdown action** occurs when the arm places the object X onto the table. □

- **FORM:** PUTDOWN(X) □

- **PRE:** HOLDING(X) □
- **ADD:** ONTABLE(X) ∧ ARMEMPTY()∧ CLEAR(X) □
- **DEL:** HOLDING(X) □
- **Constraints:** *X ≠ Table*

To do these operations, we need to use the following predicates:

- ON(X,Y) – Block x is on block Y
- ONTABLE(X) – Block X is on table
- CLEAR(X) – there is nothing on top of block X
- HOLDING(X) – The arm is holding block X.
- ARMEMPTY – The arm is holding nothing

COMPONENTS OF A PLANNING SYSTEM

In problem solving systems, it is necessary to perform each of the following functions:

- Choose the best rule to apply next based on the best available heuristic information
- Apply the chosen rule to compute the new problem state that arises from its application.
- Detect when a solution has been found
- Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.
- Detect when almost correct solution has been found and employ special techniques to make it totally correct.

i) Choosing rule to apply

The most widely used technique for selecting appropriate rules to apply is first to isolate a set of differences between the desired goal state and the current state and then to identify those rules that are relevant to reducing those differences. If several rules are found, a variety of other heuristic information can be exploited to choose among them.

ii) Applying Rules

To handle complex domains, STRIPS problem solver is used. In this approach each operation is described using three lists

- Precondition List – a list of facts which must be true for action to be executed.
- DELETE list - a list of facts that are no longer true after action is performed; *f*
- ADD list - a list of facts made true by executing the action.

Basic operations

- *stack(X,Y)*: put block X on block Y
- *unstack(X,Y)*: remove block X from block Y
- *pickup(X)*: pickup block X from the table
- *putdown(X)*: put block X on the table *f*

The **stack action** occurs when the robot arm places the object it is holding [X] on top of another object [Y].

- **FORM:** STACK(X,Y) *f*
- **PRE:** CLEAR(Y) ∧ HOLDING(X) □
- **ADD:** ARMEMPTY() ∧ ON(X,Y) ∧ CLEAR(X) □
- **DEL:** CLEAR(Y) ∧ HOLDING(X) □
- **CONSTRAINTS:** (X ≠ Y), X ≠ TABLE, Y ≠ TABLE

The **unstack action** occurs when the robot arm picks up an object X from on top of another object Y.

- **FORM:** UNSTACK(X,Y) □
- **PRE:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) ∧ CLEAR(Y) □
- **DEL:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **CONSTRAINTS:** X ≠ Y, X ≠ TABLE, Y ≠ TABLE

The **pickup action** occurs when the arm picks up an object (block) X from the table. □

- **FORM:** PICKUP(X) □
- **PRE:** ONTABLE(X) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) □

- **CONSTRAINTS:** X ≠ TABLE

The **putdown action** occurs when the arm places the object X onto the table. □

- **FORM:** PUTDOWN(X) □
- **PRE:** HOLDING(X) □
- **ADD:** ONTABLE(X) ∧ ARMEMPTY()∧ CLEAR(X) □
- **DEL:** HOLDING(X) □
- **Constraints:** *X ≠ Table*

iii) Detecting a Solution

A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transform the initial problem state in to goal state. In simple problem solving systems, a straightforward match of the state descriptions has done.

iv) Detecting dead ends

As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution.

If the search process is reasoning forward from initial state, it can prune any a path that leads to a state from which the goal state cannot be reached.

If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or because little progress is being made. In reasoning backward, each goal is decomposed in to sub goals. Each of them in turn may lead to a set of additional sub goals. Sometimes it is easy to detect that there is no way that all sub goals in a given set can be satisfied at once.

v) Repairing an almost correct solution

The problem is decomposed completely, proceed to solve sub problems separately and check that when solutions are combined, they do in fact yield a solution to the original problem. If they do not throw out the solution, look for another one.

A slightly better approach to look at the situation that results when the sequence of operations corresponding to the proposed solution is executed and to compare the situation to the desired goal. In most cases, the difference between the two will be smaller than the difference between initial state and goal. Now the problem solving can be called again and asked to find a way of eliminating this new difference. The first solution can then be combined with this second one to form a solution to the original problem.

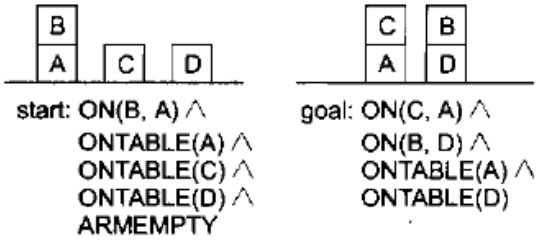
An even better way to patch up an almost correct solution is to appeal to specific knowledge about what went wrong and then apply a direct path.

A still better way to patch up incomplete solutions is not really to patch them up at all, but rather to leave them incompletely specified until the last possible moment. Then when as much information as possible is available, complete specification in such a way that no conflicts arise. This approach can be thought of as a least commitment strategy. It can be applied in variety of ways. One is to defer deciding on the order in which operations will be performed.

GOAL STACK PLANNING

One of the earliest techniques to be developed for solving compound goals that may interact was the use of a goal stack.

This approach was used by STRIPS. A single stack is maintained which contains both goals and operators.



When we begin solving this problem, the goal stack is simply

ON(C,A) ∧ON(B,D) ∧ONTABLE(A) ∧ONTABLE(D)

Separate this problem in to sub problems, one for each component of the original goal. Two of the sub-problems ONTABLE(A) and ONTABLE(D) are already true in the initial state. OTAD is the abbreviation for ONTABLE(A) ∧ ONTABLE(D)

ON(C,A)
ON(B,D)
ON(C,A) ∧ ON(B,D) ∧ OTAD

At each succeeding step of the problem solving process, the top goal on the stack will be pursued. When a sequence of operators that satisfies is found, that sequence is applied to the state description, yielding a new description. Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal. This process continues until the goal is empty.

Then as one last check, the original goal is compared to the final state derived from the application of the chosen operators. If any components of the goal are not satisfied in that state, then those unsolved parts of the goal are reinserted on to the stack and the process resumed.

ON(C,A) is replaced with the operator STACK(C,A).

STACK(C,A)
ON(B,D)
ON(C,A) ∧ ON(B,D) ∧ OTAD

The preconditions of STACK(C,A) is pushed inside the stack.

CLEAR(A)
HOLDING(C)
CLEAR(A) ∧ HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ ON(B,D) ∧ OTAD

CLEAR(A) is not true. To make CLEAR(A) true, B must be un-stacked from A. This is done by the operator UNSTACK(B,A).

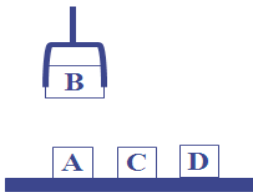
UNSTACK(B,A)
HOLDING(C)
CLEAR(A) ∧ HOLDING(C) STACK(C,A)
ON(B,D)
ON(C,A) ∧ ON(B,D) ∧ OTAD

The preconditions of UNSTACK(B,A) are pushed inside the goal stack.

ON(B,A)
CLEAR(B)
ARMEMPTY
ON(B,A) ∧ CLEAR(B) ∧ ARMEMPTY
UNSTACK(B,A)
HOLDING(C)
CLEAR(A) ∧ HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ ON(B,D) ∧ OTAD

Here, ON(B,A) is true, CLEAR(B) is true, ARMEMPTY is true. So, they are popped out from the goal stack. Thus, all the preconditions of UNSTACK(B,A) is true. So, UNSTACK(B,A) is placed in the solution list.

Solution List={ UNSTACK(B,A) }



HOLDING(C) is not true because the arm is holding B.

HOLDING(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Thus, HOLDING(C) is replaced with the operator PICKUP(C)

PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

The preconditions for PICKUP(C) are pushed inside the goal stack.

ONTABLE(C)
CLEAR(C)
ARMEMPTY
ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Here, ONTABLE(C) is true, CLEAR(C) is true. So ONTABLE(C) and CLEAR(C) are popped out from the goal stack. ARMEMPTY is not true because the arm is holding C. Thus, make the arm empty by stacking B on D. This done by the operator STACK(B.D), which replaces ARMEMPTY.

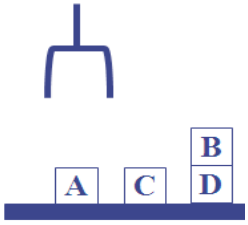
STACK(B,D)
ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

The preconditions of STACK(B,D) is pushed inside the goal stack.

CLEAR(D)
HOLDING(B)
CLEAR(D) ∧HOLDING(B)
STACK(B,D)
ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

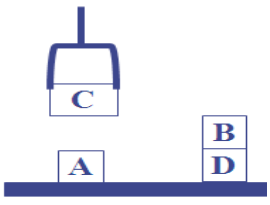
Here, CLEAR(D) and HOLDING(B) are true. This means that the preconditions of STACK(B,D) is true. Thus, CLEAR(D) and HOLDING(B) are popped out and STACK(B,D) is placed in the solution list.

Solution List={ UNSTACK(B,A), STACK(B,D)}



ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

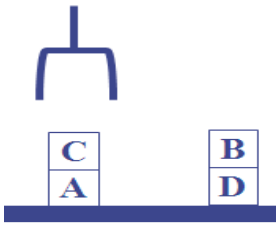
Now, ONTABLE(C), CLEAR(C) and ARMEMPTY are true. So they are popped out and PICKUP(C) is placed on the solution list.
Solution List={ UNSTACK(B,A), STACK(B,D), PICKUP(C)}



CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Here, CLEAR(A) and HOLDING(C) are true. Thus, CLEAR(A) and HOLDING(C) are popped out from the goal stack. STACK(C,A) is placed in the solution list.

Solution List={ UNSTACK(B,A), STACK(B,D), PICKUP(C), STACK(C,A)}



Thus, the goal is reached.

HIERARCHICAL PLANNING

- To solve hard problems, problem solver may have to generate long plans.
- To do this efficiently, eliminate some of the details of the problem until a solution that addresses the main issues is found. Then an attempt can be made to fill in the appropriate details.
- ABSTRIPS is an approach, which actually planned in a hierarchy of abstraction spaces, in which each of the preconditions at a lower level of abstractions were ignored.

```
(OPERATOR
    (PRECONDITIONS
        (and(...)
            (forall (w ...)...)
            (not
                (exists ...)
                (or.....)))
    (POSTCONDITIONS
        (ADD (...))
        (DELETE (...))
        (if (and (...)(...))
            (ADD (...)(...))
            (DELETE (...)(...))))))
```

Example: Suppose we want to visit a friend in Europe, but you have a limited amount of cash to spend. It makes sense to check air fare first, since finding an affordable flight will be most difficult part of the task. You should not worry about getting your drive way, planning a route to airport, or parking your car until you are sure you have a flight.

The ABSTRIPS approach to problem solving is as follows:

- First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
- Simply ignore preconditions of lower than peak criticality.
- Once this is done, use the constructed plan as the outline of a complete plan and consider preconditions at the next lowest criticality level.
- Augment the plan with operators that satisfy those preconditions.
- Again in choosing operators, ignore all preconditions whose criticality level is less than the level now being considered.
- Continue this process of considering less and less critical preconditions until all of the preconditions of the original rules have been considered.

Because this process explores entire plans at one level of detail before it looks at the lower level details of any of them. This is called **length-first search**.

REACTIVE SYSTEMS

Hierarchical planning and Non-linear planning systems are deliberative planning process, in which a plan for completing an entire task is constructed prior to action. The idea of reactive systems is to avoid planning altogether, and instead use the observable situation as a clue to which one can simply react.

It chooses actions one at a time; it does not anticipate and select an entire action sequence before it does the first thing.

- **Example**: Thermostat

The job of the thermostat is to keep temperature constant inside a room. The real thermostat uses simple pair of **situation-action rule.**

If the temperature in the room is k degrees above the desired temperature, then turn the air condition on.

If the temperature in the room is k degrees below the desired temperature, then turn the air condition off.

An intelligent system with limited resources must decide when to start thinking, when to stop thinking and when to act. Some mechanisms foe suspending plan execution is needed so that the system can turn its attention to high priority goals. Finally some situations require immediate attention and rapid action. For this reason, some deliberative planners compile out relative sub systems based on their problem solving experiences.

Advantages:

1. Robustness – operate robustly in domains that are difficult to model completely and accurately
2. Extremely responsive – attractive for real time tasks like driving and walking.

Other planning Systems

Triangle Tables: Provide a way of recording the goals that each operator is expected to satisfy as well as goals that must be true for it to execute correctly. If something unexpected happens during the execution of a plan, the table provides the information required to patch the plan.

Meta-planning: A technique for reasoning not just about the problem being solved but also about the planning process itself.

Macro-operators: Allow a planner to build new operators that represent commonly used sequence of operators.

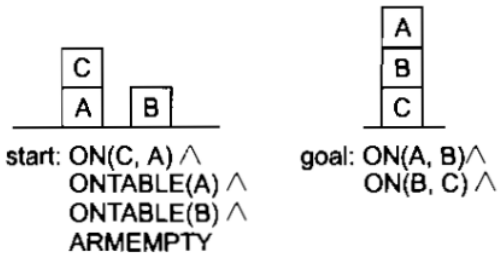
Case based planning: Re-uses old plans to make new ones.

NONLINEAR PLANNING USING CONSTRAINT POSTING

The goal stack planning method attacks problems involving conjoined goals by solving the goals one at a time, in order. A plan generated by this method contains sequence of operators for attaining the first goal, followed by the complete sequence for the second goal. But, the difficult problems cause goal interactions.

The operators used to solve one sub problem may interfere with the solution to a previous sub problem. Most sub problems require an intertwined plan in which multiple sub-problems are worked on simultaneously. Such plan is called a **nonlinear plan** because it is not composed of a linear sequence of complete sub-plans.

Let us reconsider the **SUSSMAN ANOMALY**



A good plan for the solution of this problem is following

1. Try to achieve ON(A,B) clearing block A putting block C on the table.

2. Achieve ON(B,C) by stacking block B on block C.
3. Complete ON(A,B) by stacking block A on block B.

The initial plan consists of no steps and by studying the goal state ideas for the possible steps are generated. There is no order or detail at this stage. Gradually more detail is introduced and constraints about the order of subsets of the steps are introduced until a completely ordered sequence is created.

Tweak Heuristics Using Constraint Posting

The *Tweak* planning method involves the following heuristics.

Step Addition

-- Creating new steps (GPS) for a plan.

Promotion

-- Constraining a step to go before another step.

Declobbering

-- Placing a new step between two steps to revert a precondition.

Simple Establishment

-- Assigning a value to a variable to ensure a precondition.

Separation

-- Preventing variables being assigned certain values.

In this problem means-end analysis suggests two steps with end conditions ON(A,B) and ON(B,C) which indicates the operator STACK giving the layout shown below where the operator is preceded by its preconditions and followed by its post conditions:

ON(A,B)	**ON(B,C)**
CLEAR(B)	CLEAR(C)
*HOLDING(A)	*HOLDING(B)
STACK(A,B)	STACK(B,C)
ARMEMPTY	ARMEMPTY
ON(A,B)	ON(B,C)
¬ CLEAR(B)	¬ CLEAR(C)
¬ HOLDING(A)	¬ HOLDING(B)

Many planning methods have introduced heuristics to achieve goals or preconditions. The TWEAK planning method brought all these together under one formalism. Other methods that introduced/used the following heuristics are mentioned in brackets in the following section.

CLEAR(A)	CLEAR(C)
ONTABLE(A)	ONTABLE(B)
*ARMEMPTY	*ARMEMPTY
PICKUP(A)	PICKUP(B)
¬ONTABLE(A)	¬ONTABLE(B)
¬ARMEMPTY	¬ARMEMPTY
HOLDING(A)	HOLDING(B)

In this case we need to state that a PICKUP step should precede a corresponding STACK step. That is to say

PICKUP(A) ⇐ STACK(A,B)

PICKUP(B) ⇐ STACK(B,C)

This gives four steps partially ordered and four unachieved conditions

- *CLEAR(A) -- block A is not clear in initial state.
- *CLEAR(B) -- although block B is clear in initial state STACK(A,B) with postcondition ¬CLEAR(B) might precede the step with *CLEAR(B) precondition.
- Two *ARMEMPTY -- initial state makes ARMEMPTY but PICKUP step has ¬ARMEMPTY and could again precede this step.

We can use the *promotion* heuristic to force one operator to precede another so that the postcondition of one operator STACK(A,B) does not negate the precondition CLEAR(B) of another operator PICKUP(B). This ordering is represented by

PICKUP(B) ⇐ STACK(A,B)

We can use promotion to achieve one of the ARMEMPTY preconditions:

Making PICKUP(B) precede PICKUP(A) ensures that the arm is empty and all the conditions for PICKUP(B) are met.

This is written

PICKUP(B) ⇐ PICKUP(A).

Unfortunately a postcondition of the first operator is that the arm becomes not empty, so we need to use the *declobbering* heuristic to achieve the preconditions of the second operator PICKUP(A).

Declobbering

- PICKUP(B) asserts ¬ARMEMPTY.
- But if we insert a step between PICKUP(B) and PICKUP(A) to reassert ARMEMPTY then we can achieve the precondition.
- STACK(B,C) can do this so we *post another constraint*:

PICKUP(B) ⇐ STACK(B,C) ⇐ PICKUP(A)

We still need to achieve CLEAR(A):
The appropriate operator is UNSTACK(x,A) by *step addition*. This leads to the following set of conditions

*CLEAR(x)
*ON(x,A)
*ARMEMPTY

UNSTACK(x,A)

¬ ON(x,A)
¬ ARMEMPTY
HOLDING(x)

CLEAR(A)

The variable x can be bound to the block C by the *simple establishment* heuristic since C is on A in the initial state. The preconditions CLEAR(C) and ARMEMPTY are negated by STACK(B,C) and by PICKUP(B) or PICKUP(A) however.

So we must introduce three orderings by *promotion* to ensure the operator UNSTACK(C,A).

UNSTACK(C,A) ← STACK(B,C)

UNSTACK(C,A) ← PICKUP(A)

UNSTACK(C,A) ← PICKUP(B)

Promotion involves adding a step and this clobbers one of the preconditions of PICKUP(B) viz ARMEMPTY, always a potential problem with this heuristic.
However all is not lost as there is an operator, PUTDOWN that has the required postcondition and given that the operator UNSTACK(C,A) had generated the precondition for it of HOLDING(C) we can produce an extra operator successfully

HOLDING(C)

PUTDOWN(C)

¬ HOLDING(C)
ONTABLE(C)
ARMEMPTY

This operator *declobbers* the operator PICKUP(B) yielding the sequence

UNSTACK(C,A) ⇐ PUTDOWN(C) ⇐ PICKUP(B)

This yields the final sequence:

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(B)
4. STACK (B,C)
5. PICKUP(A)
6. STACK(A,B)

Let us finish this section by looking at the formal form of the TWEAK algorithm:

Algorithm: Non Linear Planning (TWEAK)

1. Initialize S to be the set of propositions in the goal state.
2. Remove some unachieved proposition P from S.
3. Achieve P by using step addition, promotion, declobbering, simple establishment, or separation.
4. Review all the steps in the plan including any new steps introduced by step addition to see if any of their preconditions are unachieved. Add to S the new set of unachieved preconditions.

5. If s is empty, complete the plan by converting the partial order of steps in to a total order, and instantiate any variables as necessary.
6. Otherwise go to step 2.

EXPLANATION

In order for an expert system to be an effective tool, people must be able to interact with it easily. To facilitate this interaction, the expert system must have the following two capabilities in addition to ability to perform its underlying task.

Explain its reasoning: In many of the domains in which expert systems operate, people will not accept results unless they have been convinced of the accuracy of the reasoning process that produced those results. This is particularly true, for example, in medicine, where a doctor must accept ultimate responsibility for a diagnosis, even if that diagnosis was arrived at with considerable help from the program.

Acquire new knowledge and modification of old knowledge. Since expert systems derive this power from the richness of the knowledge bases they of exploit, it is extremely important that those knowledge bases be as complete and as accurate as possible. But often there exists no standard codification of that knowledge; rather it exists only inside the heads of human experts. One way to get this knowledge in to program is through interaction with human expert. Another way is to have the program learn expert behavior from raw data.

LEARNING

Learning process is the basis of knowledge acquisition process. Knowledge acquisition is the expanding the capabilities of a system or improving its performance at some specified task. The acquired knowledge may consist of various facts, rules, concepts, procedures, heuristics, formulas, relationships or any other useful information.

Rote Learning

It 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. It includes learning my imitation, simple memorization and learning by being performed.

For example we may use this type of learning when we memorize multiplication tables. In this method we store the previous computed values, for which we do not have for recomputed them later. Also we can say rote learning is one type of existing or base learning.

For example, in our childhood, we have the knowledge that “sun rises in the east”. So in our later stage of learning we can easily memorize the thing. Hence in this context, a system may simply memorize previous solutions and recall them when confronted with the same problem. Generally access of stored value must be faster than it would be to re-compute. Methods like hashing, indexing and sorting can be employed to enable this.

One drawback of rote learning is it is not very effective in a rapidly changing environment. If the environment does change then we must detect and record exactly what has changed. Also this technique must not decrease the efficiency of the system.

Learning by Taking Advice

In this process we can learn through taking advice from others. The idea of advice taking learning was proposed in early 1958 by McCarthy. In our daily life, this learning process is quite common. Right from our parents, relatives to our teachers, when we start our educational life, we take various advices from others. We know the computer programs are written by programmers. When a programmer writes a computer program he or she gives many instructions to computer to

follow, the same way a teacher gives his/her advice to his students. The computer follows the instructions given by the programmer. Hence, a kind of learning takes place when computer runs a particular program by taking advice from the creator of the program.

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

Example: (avoid(take-points me)(trick))

LEARNING IN PROBLEM SOLVING

Describes about problem solving can be performed without the aid of teacher.

Learning by Parameter Adjustment

Many programs rely on an evaluation procedure that combines information from several sources in to a single summary statistic. Pattern classification program often combine several features to determine the correct category in to which a given stimulus should be placed. In designing such programs, it is often difficult to know a priori how much weight should be attached to each feature being used. One way to finding the correct weights is to begin with some estimate of the correct settings and then to let program modify the settings on the basis of its experience. 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.

Learning with Macro-Operators

For example, suppose you are faced with the problem of getting to a downtown post office. Your solution may involve getting in your car, starting it and driving it along a certain route. Substantial planning may go in to choosing the appropriate route, but, you need not plan about how to go about starting your car. You are free to treat START-CAR as an atomic action, even though it really consists of several actions: sitting down, adjusting the mirror, inserting the key and turning the key. Sequence of actions that can be treated as a whole are called macro-operators. A MACROP is just like a regular operator except that it consists of sequence of actions, not just a single one.

Suppose we are given an initial blocks world situation in which ON(C, B) and ON(A, Table) are both true. STRIPS can achieve goal ON(A,B) by devising a plan with the four steps UNSTACK(C,B), PUTDOWN(C), PICKUP(A), STACK(A,B). STRIPS now build a MACROP with preconditions ON(C,B), ON(A, Table) and post conditions ON(C, Table), ON(A,B).

Learning by Chunking

Chunking is a process similar to macro-operators. Its computational basis is in production systems. SOAR also exploits chunking so that the performance can increase with experience. SOAR solve problem by firing productions which are stored in long term memory. When SOAR detects a useful sequence of production firings, it creates a chunk which is essentially a large production that does the work of an entire sequence of smaller ones.

SOAR learns how to place a given tile without permanently disturbing the previously placed tiles. Given the way that SOAR learns, several chunks may encode a single macro-operator, and one chunk may participate in a number of macro sequences. Chunks are generally applicable toward any goal state. Chunks learned during the initial stages of solving a problem are applicable in the later stages of the same problem solving episode. After a solution is found, the chunks remain in memory, ready for use in the next problem.

LEARNING FROM EXAMPLES: INDUCTION

- Classification is the process of assigning to a particular input, the name of a class to which it belongs.
- It is an important component of many problem solving tasks.
- Often classification is embedded inside another operation.

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

- 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.

  $$c_1t_1 + c_2t_2 + c_3t_3 + ...$$

  Each t corresponds to a value of a relevant parameter, and each c represents the weight to be attached to the corresponding t. 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 the location of cold fronts. Different functions can be written to combine these parameters to predict sunny, rainy, or snowy weather.

- Isolate a set of features that are relevant to the task domain. Define each class as 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 as things as color, length of neck, and feathers

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.**

If classes are described by scoring functions then concept learning can be done using the technique of coefficient adjustment. If classes are defined structurally, some other techniques are necessary.

Three such techniques are:

1. Winston's learning program
2. Version Spaces
3. Decision trees

1. Winston's learning program

A Blocks World Learning Example -- Winston (1975)

- The goal is to construct representation of the definitions of concepts in this domain.
- Concepts such a house - brick (rectangular block) with a wedge (triangular block) suitably placed on top of it, tent - 2 wedges touching side by side, or an arch - two non-touching bricks supporting a third wedge or brick, were learned.
- The idea of *near miss* objects -- similar to actual instances was introduced.
- Input was a line drawing of a blocks world structure.

- Input processed (see VISION Sections later) to produce a semantic net representation of the structural description of the object.

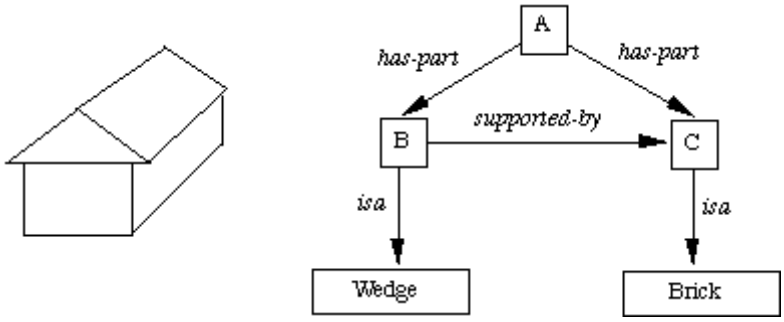


Fig. House object and semantic net

- Links in network include left-of, right-of, does-not-marry, supported-by, has-part, and isa.
- The marry relation is important -- two objects with a common touching edge are said to marry. Marrying is assumed unless does-not-marry stated.

There are three basic steps to the problem of concept formulation:

1. Select one know instance of the concept. Call this the concept definition.
2. Examine definitions of other known instance of the concept. Generalize the definition to include them.
3. Examine descriptions of near misses. Restrict the definition to exclude these.

Both steps 2 and 3 rely on comparison and both similarities and differences need to be identified.

2. Version Spaces

The goal of version space is to produce a description that is consistent with all positive examples but no negative examples in the training set. Version space works by maintaining a set of possible descriptions and evolving that set as new examples and near misses are presented. Some sort of representational knowledge is needed for examples so that we can describe exactly what the system sees in the example.

The algorithm for narrowing version space is called the candidate elimination algorithm

Algorithm: Candidate Elimination

Given: A representation language and a set of positive and negative examples expressed in the language.

Compute: a concept description that is consistent with all positive examples and none of the negative examples.

1. Initialize G to contain one element: the null description.
2. Initialize S to contain one element: the first positive example.
3. Accept a new training example.

If it is a positive example, first remove from G any descriptions that do not cover the example. Then update s set to contain the most specific set of descriptions in the version space that cover the example and the current elements of the set S.

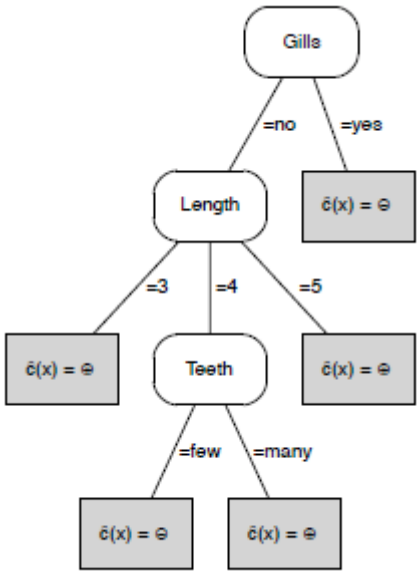
If it is a negative example, first remove S from any descriptions that cover the example. Then, update the G set to contain the most general set of descriptions in the version space that do not cover the example. That is, specialize the elements of G as little as possible so that the negative example is no longer covered by any of the elements of G.

4. If S and G are both singleton sets, then if they are identical, output their value and halt. If they are both singleton sets but they are different, then the training cases were inconsistent. Output this result and halt. Otherwise go to step 3.

3. Decision trees

A method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree or sets of if-then rules. Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.

Task: Distinguish dolphins from other similar looking fishes
Features: Length (numeric), Gills (yes| no), Beak (yes| no), Teeth (categorical — Many | Few)



EXPLANATION BASED LEARNING

An Explanation-based Learning (**EBL**) system accepts an example (i.e. a training example) and explains what it learns from the example. The **EBL** system takes only the relevant aspects of the training. This explanation is translated into particular form that a problem solving program can understand. The explanation is generalized so that it can be used to solve other problems.

PRODIGY is a system that integrates problem solving, planning, and learning methods in a single architecture. It was originally conceived by Jaime Carbonell and Steven Minton, as an AI system to test and develop ideas on the role that machine learning plays in planning and problem solving. **PRODIGY** uses the **EBL** to acquire control rules.

The **EBL** module uses the results from the problem-solving trace (ie. Steps in solving problems) that were generated by the central problem solver (a search engine that searches over a problem space). It constructs explanations using an axiomatized theory that describes both the domain and the architecture of the problem solver. The results are then translated as control rules and added to the knowledge base. The control knowledge that contains control rules is used to guide the search process effectively.

Consider the problem of learning the concept bucket. We want to generalize from a single example of a bucket. At first collect the following information.

1. **Input Examples:**

   Owner (object, X) K has part (object, Y) K is(object, Deep) K Color (Object, Green) K … … (Where Y is any thin material)

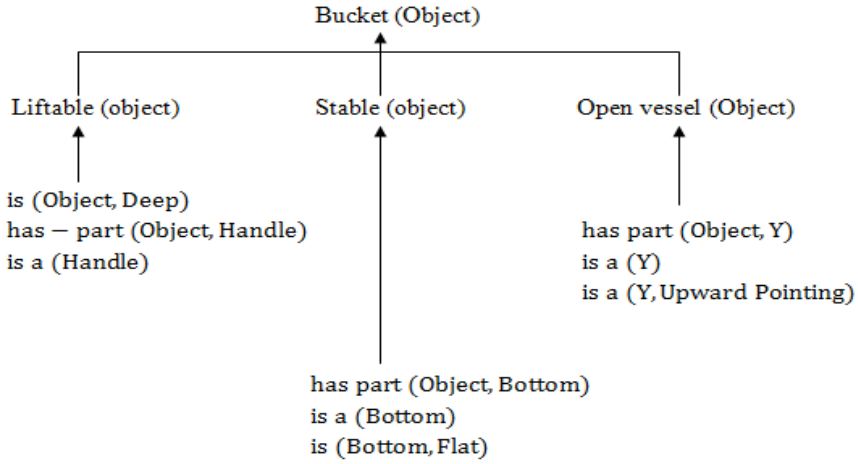
2. **Domain Knowledge:**

is (a, Deep) K has part (a, b) K is a(b, handle) ‹ liftable
(a) has part (a, b) K is a (b, Bottom) K is (b, flat) ‹ Stable
(a)
has part (a, b) K is a (b, Y) K is (b, Upward — pointing) ‹ Open — vessel (a)

3. **Goal:** Bucket

   B is a bucket if B is liftable, stable and open-vessel.

4. **Description of Concept:** These are expressed in purely structural forms like Deep, Flat, rounded etc.



GENETIC ALGORITHM

Genetic algorithms are based on the theory of natural selection and work on generating a set of random solutions and making them compete in an area where only the fittest survive. Each solution in the set is equivalent to a chromosome. 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 matting and reproduction.

Generally genetic algorithm uses three basic genetic operators like reproduction, crossover and mutation. These are combined together to evolve a new population. Starting from a random set of solutions the algorithm uses these operators and the fitness function to guide its search for the optimal solution. The fitness function guesses how good the solution in question is and provides a measure to its capability. The main advantage of the genetic algorithm formulation is that fairly accurate results may be obtained using a very simple algorithm.

Step 1: Generate the initial population.
Step 2: Calculate the fitness function of each individuals.
Step 3: Some sort of performance utility values or the fitness values are assigned to individuals.
Step 4: New populations are generated from the best individuals by the process of selection.
Step 5: Perform the crossover and mutation operation.
Step 6: Replace the old population with the new individuals.
Step 7: Perform step-2 until the goal is reached.

Applications and Advantages of Genetic Algorithm

® GA is an optimization technique guided by the principle of natural genetic systems.
® The GA is being applied to a wide range of optimization and learning problems in many domains.
® GAs solve problems using principles inspired by natural population genetics.
® GAs can provide globally optimal solutions.
® GAs use probabilistic transition rules, non-deterministic rules.

MACHINE LEARNING

Machine learning is the systematic study of algorithms and systems that improve their knowledge or performance (learn a model for accomplishing a task) with experience (from available data /examples)

Examples:

- Given an URL decide whether it is a Sports website or not
- Given that a buyer is buying a book at online store, suggest some related products for that buyer
- Given an ultrasound image of abdomen scan of a pregnant lady, predict the weight of the baby
- Given a CT scan image set, decide whether there is stenosis or not
- Given marks of all the students in a class, assign relative grades based on statistical distribution
- Given a mail received, check whether it is a SPAM
- Given a speech recording, identify the emotion of the speaker
- Given a DNA sequence, predict the promoter regions in that sequence

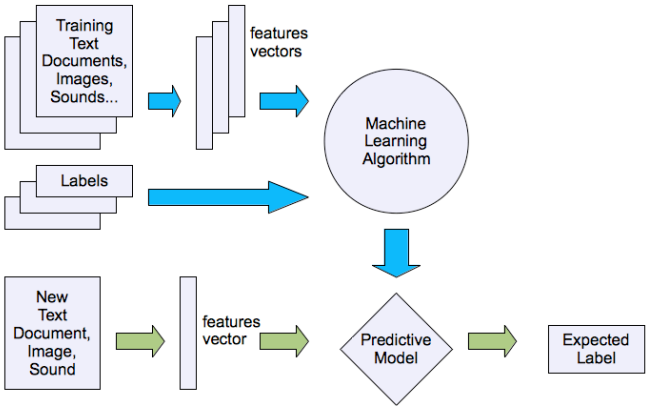
These are some examples of "Intelligent tasks" — tasks that are "easy" for humans but "extremely difficult" for a machine to achieve Artificial Intelligence is about building systems that can efficiently perform such "intelligent tasks"

One of the important aspects that enable humans to perform such intelligent tasks is their ability to learn from experiences (either supervised or unsupervised)

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system.

(i) Supervised learning

The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.



- **Prediction**
- **Classification (discrete labels),**
- **Regression (real values)**

Prediction

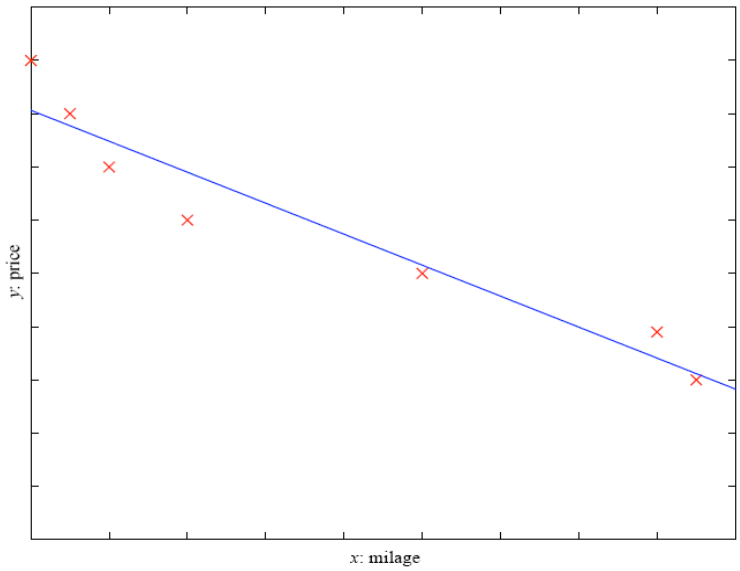
Example: Price of a used car

x : car attributes

y : price

$y = g\ (x \mid \theta\ )$

θ parameters



Classification

Example 1

Suppose you have a basket and it is filled with different kinds of fruits. Your task is to arrange them as groups. For understanding let me clear the names of the fruits in our basket.

You already learn from your previous work about the physical characters of fruits. So arranging the same type of fruits at one place is easy now. Your previous work is called as training data in data mining. You already learn the things from your train data; this is because of response variable. Response variable means just a decision variable.

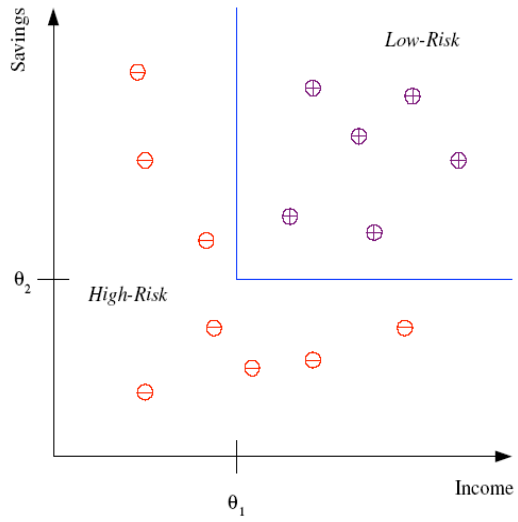
No.	SIZE	COLOR	SHAPE	FRUIT NAME
1	Big	Red	Rounded shape with a depression at the	Apple
2	Small	Red	Heart-shaped to nearly globular	Cherry
3	Big	Green	Long curving cylinder	Banana
4	Small	Green	Round to oval, Bunch shape Cylindrical	Grape

Suppose you have taken a new fruit from the basket then you will see the size, color and shape of that particular fruit. If size is Big , color is Red , shape is rounded shape with a depression at the top, you will conform the fruit name as apple and you will put in apple group.

If you learn the thing before from training data and then applying that knowledge to the test data (for new fruit), this type of learning is called as Supervised Learning.

Example 2

Credit scoring
Differentiating between low-risk and high-risk customers from their *income* and *savings*
Discriminant: IF *income* > θ_1 AND *savings* > θ_2
THEN low-risk ELSE high-risk

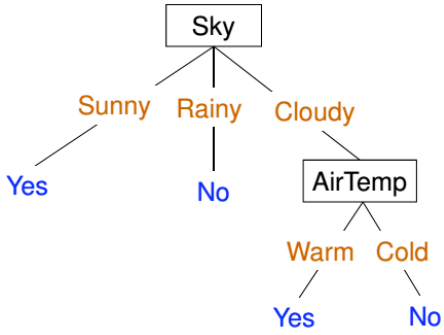


Regression

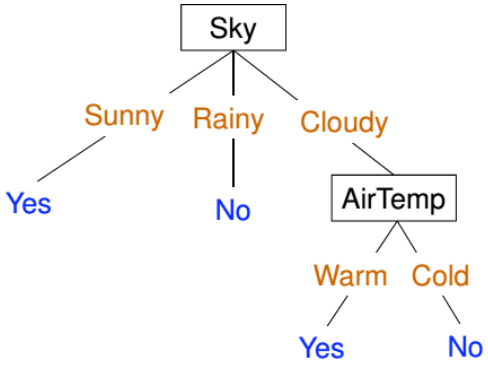
Given example pairs of heights and weights of a set of people, find a model to predict the weight of a person from her height

Decision Tree Learning
A method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree or sets of if-then rules.

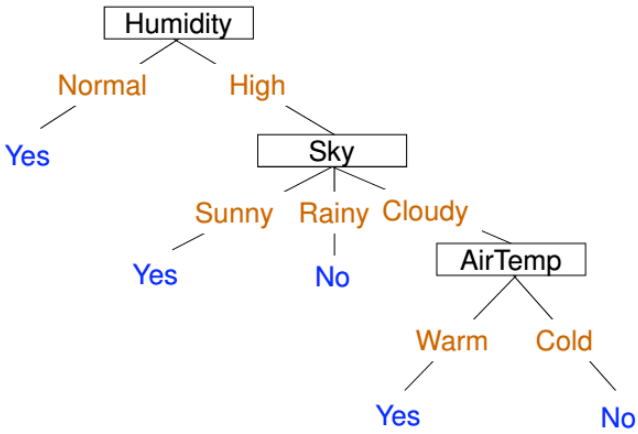
Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes
5	Cloudy	Warm	High	Weak	Cool	Same	Yes
6	Cloudy	Cold	High	Weak	Cool	Same	No



(Sky = Sunny) ∨ (Sky = Cloudy ∧ AirTemp = Warm)



7	Rainy	Warm	Normal	Weak	Cool	Same	?
8	Cloudy	Warm	High	Strong	Cool	Change	?



Information gain

- Entropy(S) = $-\ p_+ \log_2 p_+ - p_- \log_2 p_-$

  $= -\ (4/6)\log_2(4/6) - (2/6)\log_2(2/6) = 0.389 + 0.528 = 0.917$

- Gain(S, Sky)

  $= \text{Entropy}(S) - \sum_{v \in \{\text{Sunny, Rainy, Cloudy}\}} (|S_v|/|S|)\text{Entropy}(S_v)$

  $= \text{Entropy}(S) - [(3/6).\text{Entropy}(S_{\text{Sunny}}) + (1/6).\text{Entropy}(S_{\text{Rainy}}) + (2/6).\text{Entropy}(S_{\text{Cloudy}})]$

  $= \text{Entropy}(S) - (2/6).\text{Entropy}(S_{\text{Cloudy}})$

  $= \text{Entropy}(S) - (2/6)[-\ (1/2)\log_2(1/2) - (1/2)\log_2(1/2)]$

  $= 0.917 - 0.333 = 0.584$

❑ Gain(S, Water)

= Entropy(S) – $\sum_{v \in \{Warm, Cool\}}(|S_v|/|S|)Entropy(S_v)$

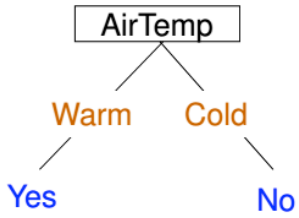
= Entropy(S) – [(3/6).Entropy(S_{Warm}) + (3/6).Entropy(S_{Cool})]

= Entropy(S) – (3/6).2.[– (2/3)$\log_2$(2/3) – (1/3)$\log_2$(1/3)]

= Entropy(S) – 0.389 – 0.528

= 0

❑ Gain(S, AirTemp) = 0.917

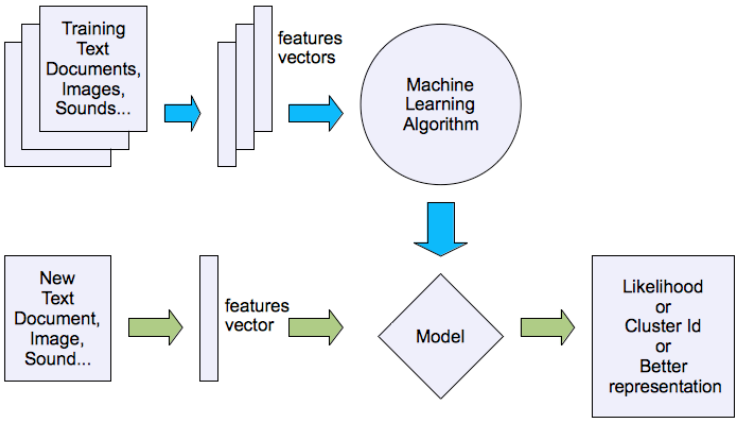


(ii) **Unsupervised learning**

Unsupervised learning, no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end.

- **Clustering**

  In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- **Probability distribution estimation**
- **Finding association (in features)**
- **Dimension reduction**
  - Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.



Example

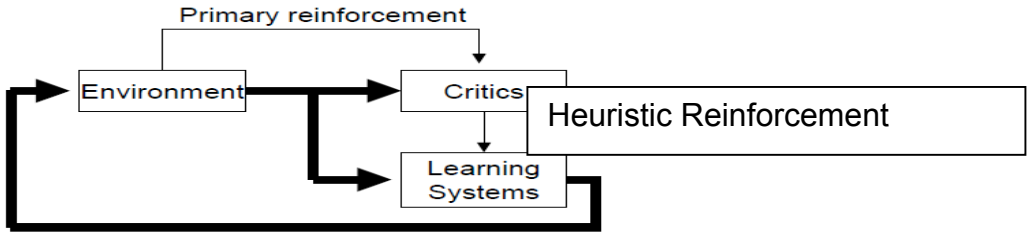
- Suppose you have a basket and it is filled with some different types fruits, your task is to arrange them as groups.

- This time you don't know anything about the fruits, honestly saying this is the first time you have seen them. You have no clue about those.
- So, how will you arrange them? What will you do first???
- You will take a fruit and you will arrange them by considering physical character of that particular fruit.
- Suppose you have considered color.
  - Then you will arrange them on considering base condition as **color.**
  - Then the groups will be something like this.
    - RED COLOR GROUP: apples & cherry fruits.
    - GREEN COLOR GROUP: bananas & grapes.
- So now you will take another physical character such as **size**.
  - RED COLOR AND BIG SIZE: apple.
  - RED COLOR AND SMALL SIZE: cherry fruits.
  - GREEN COLOR AND BIG SIZE: bananas.
  - GREEN COLOR AND SMALL SIZE: grapes.
- Job done happy ending.
- Here you did not learn anything before, means no train data and no response variable.
- This type of learning is known as **unsupervised learning.**
- **Clustering** comes under **unsupervised learning**.

(iii) Reinforcement learning

In reinforcement learning, a computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent.

- Decision making (robot, chess machine)



NEURAL NETWORK

A neural network consists of inter connected processing elements called neurons that work together to produce an output function. The output of a neural network relies on the cooperation of the individual neurons within the network to operate. Well-designed neural networks are trainable systems that can often "learn" to solve complex problems from a set of exemplars and generalize the "acquired knowledge" to solve unforeseen problems, i.e. they are self-adaptive systems. A neural network is used to refer to a network of biological neurons. A neural network consists of a set of highly interconnected entities called nodes or units. Each unit accepts a weighted set of inputs and responds with an output.

Mathematically let $I = (I_1, I_2, \ldots \ldots I_n)$ represent the set of inputs presented to the unit U. Each input has an associated weight that represents the strength of that particular connection. Let $W = (W_1, W_2, \ldots \ldots W_n)$ represent the weight vector corresponding to the input vector X. By applying to
V, these weighted inputs produce a net sum at U given by

$$S = SUM (W_i \times I_i)$$

FEATURES OF ARTIFICIAL NETWORK (ANN)

ANN is just a parallel computational system consisting of many simple processing elements connected together in a specific way in order to perform a particular task. There are some important features of artificial networks as follows.

(1) Artificial neural networks are extremely powerful computational devices (Universal computers).

(2) ANNs are modeled on the basis of current brain theories, in which information is represented by weights.

(3) ANNs have massive parallelism which makes them very efficient.

(4) They can learn and generalize from training data so there is no need for enormous feats of programming.

(5) Storage is fault tolerant i.e. some portions of the neural net can be removed and there will be only a small degradation in the quality of stored data.

(6) They are particularly fault tolerant which is equivalent to the "graceful degradation" found in biological systems.

(7) Data are naturally stored in the form of associative memory which contrasts with conventional memory, in which data are recalled by specifying address of that data.

(8) They are very noise tolerant, so they can cope with situations where normal symbolic systems would have difficulty.

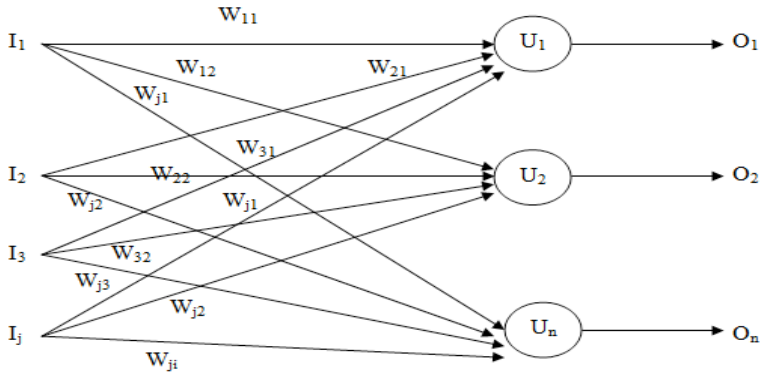
(9) In practice, they can do anything a symbolic/ logic system can do and more.

(10) Neural networks can extrapolate and **intrapolate** from their stored information. The neural networks can also be trained. Special training teaches the net to look for significant features or relationships of data.

TYPES OF NEURAL NETWORKS

Single Layer Network

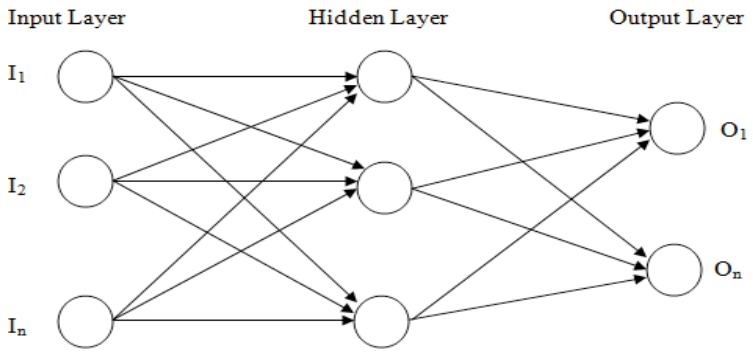
A single layer neural network consists of a set of units organized in a layer. Each unit U_n receives a weighted input I_j with weight W_{jn}. Figure shows a single layer neural network with j inputs and outputs.



Single Layer neural Network

Multilayer Network

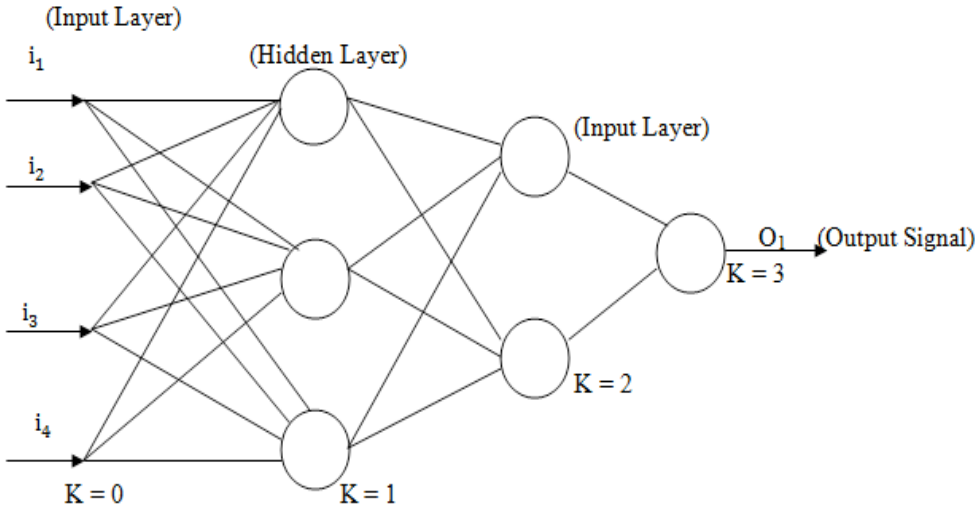
A multilayer network has two or more layers of units, with the output from one layer serving as input to the next. Generally in a multilayer network there are 3 layers present like, input layer, output layer and hidden layer. The layer with no external output connections are referred to as hidden layers. A multilayer neural network structure is given in figure.



A multilayer neural network

Feed Forward neural network

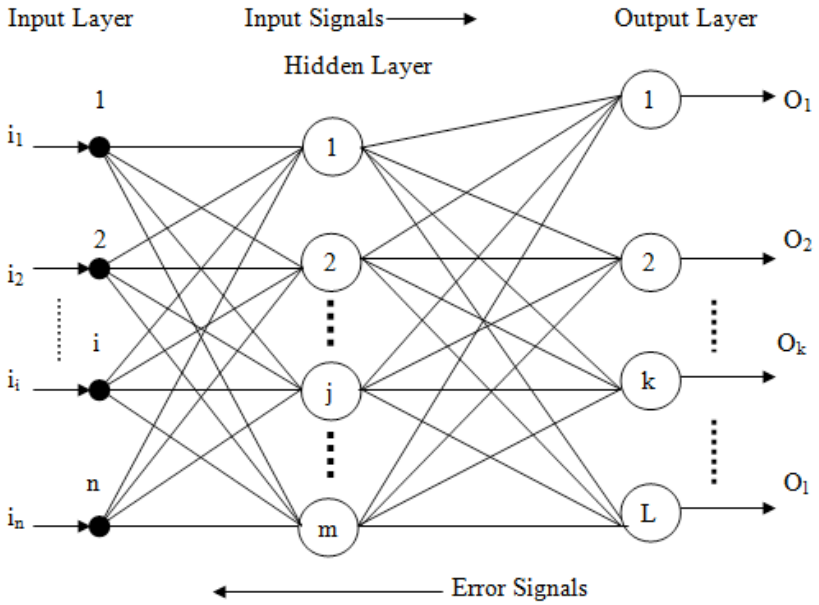
In this network, the information moves in only one direction, forward from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network. In other way we can say the feed forward neural network is one that does not have any connections from output to input. All inputs with variable weights are connected with every other node. A single layer feed forward network has one layer of nodes, whereas a multilayer feed forward network has multiple layers of nodes. The structure of a feed forward multilayer network is given in figure.



Multilayer Feed Forward Neural Network

Back Propagation neural network

Multilayer neural networks use a most common technique from a variety of learning technique, called the back propagation algorithm. In back propagation neural network, the output values are compared with the correct answer to compute the value of some predefined error function. By various techniques the error is then fed back through the network. Using this information, the algorithms adjust the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles the network will usually converge to some state where the error of the calculation is small.



The goal of back propagation, as with most training algorithms, is to iteratively adjust the weights in the network to produce the desired output by minimizing the output error. The algorithm's goal is to solve credit assignment problem. Back propagation is a gradient-descent approach in that it uses the minimization of first-order derivatives to find an optimal solution. The standard back propagation algorithm is given below.

Step1: Build a network with the choosen number of input, hidden and output u n i t s .
Step2: Initialize all the weights to low random values.
Step3: Randomly, choose a single training pair.
Step4: Copy the input pattern to the input layer.
Step5: Cycle the network so that the activation from the inputs generates the activations in the hidden and output layers.
Step6: Calculate the error derivative between the output activation and the final o u t p u t .
Step7: Apply the method of back propagation to the summed products of the weights and errors in the output layer in order to calculate the error in the hidden units.
Step8: Update the weights attached the each unit according to the error in that unit, the output from the unit below it and the learning parameters, until the error is sufficiently low

ADAPTIVE LEARNING

Adaptive learning refers broadly to a learning process where the content taught or the way such content is presented changes or "adapts" based on the responses of the individual student.

Adaptive learning is an educational method which uses computers as interactive teaching devices, and to orchestrate the allocation of human and mediated resources according to the unique needs of each learner. Computers adapt the presentation of educational material according to students' learning needs, as indicated by their responses to questions, tasks and experiences. The technology encompasses aspects derived from various fields of study including computer science, education, psychology, and brain science.

Adaptive learning has been implemented in several kinds of educational systems such as adaptive educational hypermedia, intelligent tutoring systems, Computerized adaptive testing, and computer-based pedagogical agents, among others.

The **goal of an** adaptive learning system is to personalize instruction in order to improve or accelerate a student's performance again.

Adaptive learning systems have traditionally been divided into separate components or 'models'.

- Expert model - The model with the information which is to be taught
- Student model - The model which tracks and learns about the student
- Instructional model - The model which actually conveys the information

Expert model

The expert model stores information about the material which is being taught. This can be as simple as the solutions for the question set but it can also include lessons and tutorials and, in more sophisticated systems, even expert methodologies to illustrate approaches to the questions.

Adaptive learning systems which do not include an expert model will typically incorporate these functions in the instructional model.

Student model

Student model algorithms have been a rich research area over the past twenty years. The simplest means of determining a student's skill level is the method employed in CAT (Computerized adaptive testing). In CAT, the subject is presented with questions that are selected based on their level of difficulty in relation to the presumed skill level of the subject. As the test proceeds, the computer adjusts the subject's score based on their answers, continuously fine-tuning the score by selecting questions from a narrower range of difficulty.

An algorithm for a CAT-style assessment is simple to implement. A large pool of questions is amassed and rated according to difficulty, through expert analysis, experimentation, or a combination of the two. The computer then performs what is essentially a binary search, always giving the subject a question which is half way between what the computer has already determined to be the subject's maximum and minimum possible skill levels. These levels are then adjusted to the level of the difficulty of the question, reassigning the minimum if the subject answered correctly, and the maximum if the subject answered incorrectly. Obviously, a certain margin for error has to be built in to allow for scenarios where the subject's answer is not indicative of their true skill level but simply coincidental. Asking multiple questions from one level of difficulty greatly reduces the probability of a misleading answer, and allowing the range to grow beyond the assumed skill level can compensate for possible misevaluations.

Instructional model

The instructional model generally looks to incorporate the best educational tools that technology has to offer (such as multimedia presentations) with expert teacher advice for presentation methods. The level of sophistication of the instructional model depends greatly on the level of sophistication of the student model. In a CAT-style student model, the instructional model will simply rank lessons in correspondence with the ranks for the question pool. When the student's level has been satisfactorily determined, the instructional model provides the appropriate lesson. The more advanced student models which assess based on concepts need an instructional model which organizes its lessons by concept as well. The instructional model can be designed to analyze the collection of weaknesses and tailor a lesson plan accordingly.

When the incorrect answers are being evaluated by the student model, some systems look to provide feedback to the actual questions in the form of 'hints'. As the student makes mistakes, useful suggestions pop up such as "look carefully at the sign of the number". This too can fall in the domain of the instructional model, with generic concept-based hints being offered based on concept weaknesses, or the hints can be question-specific in which case the student, instructional, and expert models all overlap.