EXPERT SYSTEMS

What is an expert system?

According to Durkin, an expert system is "A computer program designed to model the problem solving ability of a human expert". The aspects of human experts that expert systems model are the experts:

1. Knowledge
2. Reasoning

Some traits that characterize experts are:

They possess specialized knowledge in a certain area
They possess experience in the given area
They can provide, upon elicitation, an explanation of their decisions
The have a skill set that enables them to translate the specialized knowledge gained through experience into solutions

Famous expert systems, which served to define the paradigms for the current expert systems.

Dendral (1960's)

Dendral was one of the pioneering expert systems. It was developed at Stanford for NASA to perform chemical analysis of Martian soil for space missions.

MYCIN (mid 70s)

MYCIN was developed at Stanford to aid physicians in diagnosing and treating patients with a particular blood disease.

R1/XCON (late 70's)

R1/XCON is also amongst the most cited expert systems. It was developed by DEC (Digital Equipment Corporation), as a computer configuration assistant.

Roles of an expert system

An expert system may take two main roles, relative to the human expert. It may replace the expert or assist the expert

Replacement of expert

In such cases replacement of an expert may be the only feasible option. Also, in cases where an expert cannot be available at a particular geographical location e.g. volcanic areas, it is expedient to use an expert system as a substitute. An example of this role is a France based oil exploration company that maintains a number of oil wells.

Assisting expert

Assisting an expert is the most commonly found role of an ES. The goal is to aid an expert in a routine tasks to increase productivity, or to aid in managing a complex situation by using an expert system that

may itself draw on experience of other (possibly more than one) individuals. XCON is an example of how an ES can assist an expert.

How are expert systems used?

Expert systems may be used in a host of application areas including diagnosis, interpretation, prescription, design, planning, control, instruction, prediction and simulation.

Control applications

In control applications, ES are used to adaptively govern/regulate the behavior of a system, e.g. controlling a manufacturing process, or medical treatment.

An example of such a
system is VM

Design

ES are used for design applications to configure objects under given design constraints,

Diagnosis and Prescription

An ES can serve to identify system malfunction points. To do this it must have knowledge of possible faults as well as diagnosis methodology extracted from technical experts, e.g. diagnosis based on patient's symptoms

Instruction and Simulation

ES may be used to guide the instruction of a student in some topic. Tutoring applications include GUIDON

Simulation

ES can be used to model processes or systems for operational study, or for use along with tutoring applications

Interpretation

'Producing an understanding of situation from given information'. This ES provides financial assistance for a commercial bank. It looks at a large number of transactions and identifies irregularities in transaction trends.

Planning and prediction

ES may be used for planning applications, e.g. recommending steps for a robot to carry out certain steps, cash management planning.

Appropriate domains for expert systems

Can the problem be effectively solved by conventional programming?

□□Is the domain well-bounded?

□□What are the practical issues involved? Is some human expert willing to cooperate? Is the expert's knowledge especially uncertain and heuristic

The expert has the following:

□□Focused area of expertise

□□Specialized Knowledge (Long-term Memory, LTM)

□□Case facts (Short-term Memory, STM)

□□Reasons with these to form new knowledge

□□Solves the given problem

Expert system structure

Knowledge Base

The knowledge base is the part of an expert system that contains the domain knowledge, the power of an ES lies to a large extent in its richness of knowledge i.e.

- Problem facts, rules
- Concepts
- Relationships

Working memory

The working memory is the 'part of the expert system that contains the problem facts that are discovered during the session' according to Durkin. One session in the working memory corresponds to one consultation. During a consultation:

- User presents some facts about the situation.
- These are stored in the working memory.
- Using these and the knowledge stored in the knowledge base, new information is inferred and also added to the working memory.

Inference Engine

The inference engine can be viewed as the processor in an expert system that matches the facts contained in the working memory with the domain knowledge contained in the knowledge base, to draw conclusions about the problem.

Explanation facility

The explanation facility is a module of an expert system that allows transparency of operation, by providing an explanation of how it reached the conclusion

Characteristics of expert systems

ES have an explanation facility.

An expert system is different from conventional programs in the sense that program control and knowledge are separate. Besides these properties, an expert system also possesses expert knowledge in that it embodies expertise of human expert. We have also seen that an ES reasons heuristically. Lastly, an expert system, like a human expert makes mistakes, but that is tolerable if we can get the expert system to perform at least as well as the human expert it is trying to emulate.

Programming vs. knowledge engineering

Conventional programming is a sequential, three step process: Design, Code,

Debug. Knowledge engineering, which is the process of building an expert system, also involves assessment, knowledge acquisition, design, testing, documentation and maintenance. Conventional programming focuses on solution, while ES programming focuses on problem. Unlike traditional programs, you don't just program an ES and consider it 'built'. It grows as you add new knowledge. Once framework is made, addition of knowledge dictates growth of ES.

People involved in an expert system project

Domain Expert

A domain expert is 'A person who posses the skill and knowledge to solve a specific problem in a manner superior to others' (Durkin).

Knowledge Engineer

A knowledge engineer is ‘a person who designs, builds and tests an Expert System’ (Durkin).

End-user

The end users are the people who will use the expert system. Correctness, usability and clarity are important ES features for an end user.

Inference mechanisms

Guided by this intuition, we have two formal inference mechanisms; forward and backward chaining

Forward Chaining

Let’s look at how a doctor goes about diagnosing a patient. He asks the patient for symptoms and then infers diagnosis from symptoms. Forward chaining is based on the same idea.

Approach

1. Add facts to working memory (WM)
2. Take each rule in turn and check to see if any of its premises match the facts in the WM
3. When matches found for all premises of a rule, place the conclusion of the rule in WM.
4. Repeat this process until no more facts can be added. Each repetition of the process is called a pass.

Issues in forward chaining

Undirected search

The forward chaining inference engine infers all possible facts from the given facts. It has no way of distinguishing between important and unimportant facts. Therefore, equal time spent on trivial evidence as well as crucial facts.

Conflict resolution

Another important issue is conflict resolution. This is the question of what to do when the premises of two rules match the given facts. Which should be fired first? If we fire both, they may add conflicting facts

Conflict resolution strategies

Fire first rule in sequence (rule ordering in list).

Assign rule priorities (rule ordering by importance).

More specific rules (more premises) are preferred over general rules.

Prefer rules whose premises were added more recently to WM (timestamping).

Parallel Strategy (view-points).

This allows us to maintain multiple view-points on the argument concurrently

Backward chaining

Backward chaining is an inference strategy that works backward from a hypothesis to a proof. You begin with a hypothesis about what the situation might be. Then you prove it using given facts, e.g. a doctor may suspect some disease and proceed by inspection of symptoms. In backward chaining terminology, the hypothesis to prove is called the goal.

Forward vs. backward chaining

Backward chaining is more focused and tries to avoid exploring unnecessary paths of reasoning. Forward chaining, on the other hand is like an exhaustive search.

Design of expert systems

We will now look at software engineering methodology for developing practical

ES. **The general stages of the expert system** development lifecycle or ESDLC are

- Feasibility study
- Rapid prototyping
- Alpha system (in-house verification)
- Beta system (tested by users)
- Maintenance and evolution

Linear model

The main phases of the linear sequence are

- Planning
- Knowledge acquisition and analysis
- Knowledge design
- Code
- Knowledge verification
- System evaluation

Planning phase

This phase involves the following steps

- Feasibility assessment
- Resource allocation
- Task phasing and scheduling
- Requirements analysis

Knowledge acquisition

The main steps in this phase are

- Knowledge acquisition from expert
- Define knowledge acquisition strategy (consider various options)
- Identify concrete knowledge elements
- Group and classify knowledge. Develop hierarchical representation where possible.
- Identify knowledge source, i.e. expert in the domain

Knowledge acquisition techniques

- Knowledge elicitation by interview
- Brainstorming session with one or more experts.
- Electronic brainstorming
- On-site observation
- Documented organizational expertise, e.g. troubleshooting manuals

Knowledge elicitation

Getting knowledge from the expert is called knowledge elicitation vs. the broader term knowledge acquisition. Elicitation methods may be broadly divided into:

- Direct Methods

o Interviews

Indirect methods
o Questionnaire

Knowledge analysis

The goal of knowledge analysis is to analyze and structure the knowledge gained during the knowledge acquisition phase. The key steps to be followed during this stage are

Identify specific knowledge elements, at the level of concepts, entities, etc.
From the notes taken during the interview sessions, extract specific info

Inference networks

Inference networks encode the knowledge of rules and strategies.

Flowcharts

Flow charts also capture knowledge of strategies. They may be used to represent a sequence of steps that depict the order of application of rule sets. Try making a flow chart that depicts the following strategy.

Knowledge design

After knowledge analysis is done, we enter the knowledge design phase. At the end of this phase, we have

Knowledge definition
Detailed design
Decision of how to represent knowledge
o Rules and Logic
o Frames
Decision of a development tool.
Internal fact structure
Mock interface

Code

This phase occupies the least time in the ESDLC. It involves coding, preparing test cases, commenting code, developing user's manual and installation guide. At the end of this phase the system is ready to be tested.

CLIPS *(Final Term Syllabus Starts from here)*

CLIPS stands for C Language Integrated Production System. CLIPS is an expert system tool which provides a complete environment for the construction of rule and object based expert systems.

Entering and Exiting CLIPS

When you start executable, you will see prompt
CLIPS>
Commands can be entered here
To leave CLIPS, enter
(exit)

All commands use ( ) as delimiters.

Fields

Fields are the main types of fields/tokens that can be used with clips. They can be:

1. Numeric fields: consist of sign, value and exponent
   - Float .e.g. 3.5e-10
   - Integer e.g. -1 , 3
2. Symbol: ASCII characters, ends with delimiter. e.g. family
3. String: Begins and ends with double quotation marks, "Ali is Ahmed's brother"

Remember that CLIPS is case sensitive

The Deftemplate construct

The Deftemplate construct defines a relation's structure
(deftemplate <relation-name> [<optional comment>] <slot-definition>
e.g.
CLIPS> (deftemplate father "Relation father"
(slot fathersName)
(slot sonsName))

Adding facts

Facts are added in the predicate format. The deftemplate construct is used to inform CLIPS of the structure of facts. The set of all known facts is called the fact list. To add facts to the fact list, use the assert command, e.g.
Facts to add:
man(ahmed)
father(ahmed, belal)
brother(ahmed, chand)
CLIPS> (assert (man (name "Ahmed")))
CLIPS>(assert (father (fathersName "Ahmed") (sonsName "Belal")))

Viewing fact list

After adding facts, you can see the fact list using command: (facts). You will see that a fact index is assigned to each fact, starting with 0. For long fact lists, use
the format
(facts [<start> [<end>]])
For example:
(facts 1 10) lists fact numbers 1 through 10

Removing facts

The retract command is used to remove or retract facts. For example:
(retract 1) removes fact 1

Modifying and duplicating facts

To modify the fathers name slot, enter the following:
CLIPS> (modify 2 (fathersName "Ali Ahmed"))

The WATCH command

The WATCH command is used for debugging programs. It is used to view the assertion and modification of facts. The command is

CLIPS> (watch facts)

The DEFFACTS construct

These are a set of facts that are automatically asserted when the (reset) command is used, to set the working memory to its initial state. For example:

```
CLIPS> (deffacts myFacts “My known facts
( man ( name “Ahmed” ) )
(father
(fathersName“Ahmed”)
(sonsName “Belal”) ) )
```

The Components of a rule

The Defrule construct is used to add rules. Before using a rule the component facts need to be defined. For example, if we have the rule

IF Ali is Ahmed’s father

THEN Ahmed is Ali’s son

We enter this into CLIPS using the following construct:

```
;Rule header
(defrule isSon “An example rule”

; Patterns
(father (fathersName “ali”) (sonsName “ahmed”)
;THEN
=>
;Actions
(assert (son (sonsName “ahmed”) (fathersName “ali”)))
)
```

CLIPS attempts to match the pattern of the rules against the facts in the fact list. If all patterns of a rule match, the rule is activated, i.e. placed on the agenda.

Agenda driven control and execution

The agenda is the list of activated rules. We use the run command to run theagenda. Running the agenda causes the rules in the agenda to be fired.

CLIPS>(run)

Displaying the agenda

To display the set of rules on the agenda, enter the command

(agenda)

Watching activations and rules

You can watch activations in the agenda by entering

(watch activations)

You can watch rules firing using

(watch rules)

Clearing all constructs

(clear) clears the working memory

The PRINTOUT command

Instead of asserting facts in a rule, you can print out messages using

(printout t "Ali is Ahmed's son" crlf)

The SET-BREAK command

This is a debugging command that allows execution of an agenda to halt at a specified rule (breakpoint)

(set-break isSon)

Once execution stops, run is used to resume it again.

(remove-break isSon) is used to remove the specified breakpoint.

Use (show-breaks) to view all breakpoints.

Loading and saving constructs

Commands cannot be loaded from a file; they have to be entered at the command prompt. However constructs like deftemplate, deffacts and defrules

can be loaded from a file that has been saved using .clp extension. The command to load the file is:

(load "filename.clp")

Pattern matching

Variables in CLIPS are preceded by ?, e.g.

?speed

?name

Variables are used on left hand side of a rule. They are bound to different values and once bound may be referenced on the right hand side of a rule. Multi-field wildcard variables may be bound to one or more field of a pattern. They are preceded by $? e.g. $?name will match to entire name(last, middle and first)

Example 1

```
;This is a comment, anything after a semicolon is a comment
;Define initial facts
(deffacts startup (animal dog) (animal cat) (animal duck) (animal turtle)(animal horse) (warm-
blooded dog) (warm-blooded cat) (warm-blooded duck) (lays-eggs duck) (lays-eggs turtle) (child-
of dog puppy) (child-of cat kitten) (child-of turtle hatchling))

;Define a rule that prints animal names
(defrule animal (animal ?x) => (printout t "animal found: " ?x crlf))

;Define a rule that identifies mammals
(defrule mammal
  (animal ?name)
  (warm-blooded ?name)
  (not (lays-eggs ?name))
  =>
  (assert (mammal ?name))
  (printout t ?name " is a mammal" crlf))

;Define a rule that adds mammals
(defrule mammal2
  (mammal ?name)
  (child-of ?name ?young)
  =>
  (assert (mammal ?young))
  (printout t ?young " is a mammal" crlf))
```

```
;Define a rule that removes mammals from fact list
;(defrule remove-mammals
;  ?fact <- (mammal ?)
;  =>
;  (printout t "retracting " ?fact crlf)
;  (retract ?fact))
;

;Define rule that adds child's name after asking user
(defrule what-is-child
      (animal ?name)
      (not (child-of ?name ?))
=>
      (printout t "What do you call the child of a " ?name "?")
      (assert (child-of ?name (read))))
```

Example 2

```
;OR example
;note: CLIPS operators use prefix notation
(deffacts startup (weather raining))

(defrule take-umbrella
  (or (weather raining)
     (weather snowing))
  =>
  (assert (umbrella required)))
```

HANDLING UNCERTAINTY WITH FUZZY SYSTEMS

Introduction

Ours is a vague world. We humans, talk in terms of 'maybe', 'perhaps', things which cannot be defined with cent percent authority. But on the other hand, conventional computer programs cannot understand natural language as computers cannot work with vague concepts. Statements such as: "Umar is tall", are difficult for computers to translate into definite rules. On the other hand, "Umar's height is 162 cm", doesn't explicitly state whether Umar is tall or short.

Classical sets

A classical set is a container, which wholly includes or wholly excludes any given Element. It was Aristotle who came up with the 'Law of the Excluded Middle', which states that any element X, must be either in set A or in set not-A. It cannot be in both. And these two sets, set A and set not-A should contain the entire universe between them. This is a binary classification system, in which everything must be asserted or denied.

Fuzzy sets

Fuzzy sets, unlike classical sets, do not restrict themselves to something lying wholly in either set A or in set not-A. They let things sit on the fence, and are thus closer to the human world. Fuzzy set is a multi-valued function (not binary)

Fuzzy Logic

Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth -- truth values between "completely true" and "completely false".

Dr. Lotfi Zadeh of UC/Berkeley introduced it in the 1960's as a means to model the uncertainty of natural languages. He was faced with a lot of criticism but today the vast number of **fuzzy logic applications** speak for themselves:

- Self-focusing cameras
- Washing machines that adjust themselves according to the dirtiness of the clothes
- Automobile engine controls
- Anti-lock braking systems
- Color film developing systems
- Subway control systems
- Computer programs trading successfully in financial markets

Fuzzy logic represents partial truth

Any statement can be fuzzy. The tool that fuzzy reasoning gives is the ability to reply to a yes-no question with a not-quite-yes-or-no answer. How does it work? Reasoning in fuzzy logic is just a matter of generalizing the familiar yes-no (Boolean) logic. If we give "true" the numerical value of 1 and "false" the numerical value of 0, we're saying that fuzzy logic also permits in between values like 0.2 and 0.7453. “In fuzzy logic, the truth of any statement becomes matter of degree”

Boolean versus fuzzy

On the other hand, in fuzzy logic, you can define any function represented by any mathematical shape. The output of the function can be discreet or continuous. In short, fuzzy logic lets us define more realistically the true functions that define real world scenarios

Membership Function (R)

The degree of truth that we have been talking about, is specifically driven out by a function called the membership function. It can be any function ranging from a simple linear straight line to a complicated spline function or a polynomial of a higher degree.

Some characteristics of the membership functions are:

- It is represented by the Greek symbol 1
- Truth values range between 0.0 and 1.0
    - o Where 0.0 normally represents absolute falseness
    - o And 1.0 represent absolute truth

Fuzzy vs. probability

It’s important to distinguish at this point the difference between probability and fuzzy, as both operate over the same range [0.0 to 1.0]. To understand their differences lets take into account the following case, where Amber is a 20 years
old girl.

1OLD(Amber) = 0.2

In probability theory:

There is a 20% chance that Amber belongs to the set of old people, there’s an 80% chance that she doesn’t belong to the set of old people.

In fuzzy terminology:

Amber is definitely not old or some other term corresponding to the value 0.2. But there are certainly no chances involved, no guess work left for the system to classify Amber as young or old.

Logical and fuzzy operators

AND, OR and NOT operators; for fuzzy systems we needed the exact operators which would act exactly the same way when given the extreme values of 0 and 1, and that would in addition also act on other real numbers between the ranges of 0.0 to 1.0. If we choose min (minimum) operator in place for AND, we get the same output, similarly max (maximum) operator replaces OR, and 1-A replaces NOT of A.

Fuzzy set representation

Usually a triangular graph is chosen to represent a fuzzy set, with the peak around the mean, which is true in most real world scenarios, as majority of the population lies around the average height. It's also an approximation of the Gaussian curve, which is a more general function in some aspects.

How is the consequent affected by the antecedent? The consequent specifies that a fuzzy set be assigned to the output. The implication function then modifies that fuzzy set to the degree specified by the antecedent. The most common ways to modify the output fuzzy set are truncation using the min function

(where the fuzzy set is "chopped off‘).

Fuzzy rules

Fuzzify inputs: Resolve all fuzzy statements in the antecedent to a degree of membership between 0 and 1. If there is only one part to the antecedent, this is the degree of support for the rule.

Apply fuzzy operator to multiple part antecedents: If there are multiple parts to the antecedent, apply fuzzy logic operators and resolve the antecedent to a single number between 0 and 1. This is the degree of support for the rule.

Apply implication method: Use the degree of support for the entire rule to shape the output fuzzy set. The consequent of a fuzzy rule assigns an entire fuzzy set to the output. This fuzzy set is represented by a membership function that is chosen to indicate the qualities of the consequent. If the antecedent is only partially true, (i.e., is assigned a value less than 1), then the output fuzzy set is truncated according to the implication method. In general, one rule by itself doesn't do much good. What's needed are two or more rules that can play off one another. The output of each rule is a fuzzy set. The output fuzzy sets for each rule are then aggregated into a single output fuzzyset. Finally the resulting set is defuzzified, or resolved to a single number.

Fuzzy inference system

Fuzzy inference system (FIS) is the process of formulating the mapping from a given input to an output using fuzzy logic. This mapping then provides a basis from which decisions can be made, or patterns discerned

Fuzzy inference systems have been successfully applied in fields such as automatic control, data classification, decision analysis, expert systems, and computer vision. Because of its multidisciplinary nature, fuzzy inference systems are associated with a number of names, such as fuzzy-rule-based systems, fuzzy expert systems, fuzzy modeling, fuzzy associative memory, fuzzy logic controllers, and simply (and ambiguously !!) fuzzy systems.

Mamdani's fuzzy inference method is the most commonly seen fuzzy methodology. Mamdani's method was among the first control systems built using fuzzy set theory. It was proposed in 1975 by

Ebrahim Mamdani as an attempt to control a steam engine and boiler combination by synthesizing a set of linguistic control rules obtained from experienced human operators.

Five parts of the fuzzy inference process

- Fuzzification of the input variables
- Application of fuzzy operator in the antecedent (premises)
- Implication from antecedent to consequent
- Aggregation of consequents across the rules
- Defuzzification of output

Fuzzify Inputs

The first step is to take the inputs and determine the degree to which they belong to each of the appropriate fuzzy sets via membership functions. The input is always a crisp numerical value limited to the universe of discourse of the input variable (in this case the interval between 0 and 10) and the output is a fuzzy degree of membership in the qualifying linguistic set (always the interval between 0 and 1). Fuzzification of the input amounts to either a table lookup or a function evaluation.

Apply fuzzy operator

Once the inputs have been fuzzified, we know the degree to which each part of the antecedent has been satisfied for each rule. If the antecedent of a given rule has more than one part, the fuzzy operator is applied to obtain one number that represents the result of the antecedent for that rule. This number will then be applied to the output function. The input to the fuzzy operator is two or more membership values from fuzzified input variables. The output is a single truth value.

Apply implication method

Before applying the implication method, we must take care of the rule's weight. Every rule has a weight (a number between 0 and 1), which is applied to the number given by the antecedent. Generally this weight is 1 (as it is for this example) and so it has no effect at all on the implication process. From time to time you may want to weigh one rule relative to the others by changing its weight value to something other than 1. Once proper weightage has been assigned to each rule, the implication method is implemented. A consequent is a fuzzy set represented by a membership function, which weighs appropriately the linguistic characteristics that are attributed to it. The consequent is reshaped using a function associated with the antecedent (a single number). The input for the implication process is a single number given by the antecedent, and the output is a fuzzy set. Implication is implemented for each rule. We will use the min (minimum) operator to perform the implication, which truncates the output fuzzy set.

Aggregate all outputs

Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are combined into a single fuzzy set. Aggregation only occurs once for each output variable, just prior to the fifth and final step, defuzzification. The input of the aggregation process is the list of truncated output functions returned by the implication process for each rule. The output of the aggregation process is one fuzzy set for each output variable.

Defuzzify

The input for the defuzzification process is a fuzzy set (the aggregate output fuzzy set) and the output is a single number. As much as fuzziness helps the rule evaluation during the intermediate steps, the final desired output for each variable is generally a single number. However, the aggregate of a fuzzy set encompasses a range of output values, and so must be defuzzified in order to resolve a single output value from the set

Perhaps the most popular **defuzzification method** is the centroid calculation, which returns the center of area under the curve. There are other methods in practice: centroid, bisector, middle of maximum (the average of the maximum value of the output set), largest of maximum, and smallest of maximum.

INTRODUCTION TO LEARNING

What is learning ?

Learning can be described as normally a relatively permanent change that occurs in behavior as a result of experience.

"Learning denotes changes in a system that ... enables a system to do the same task more efficiently the next time." --Herbert Simon

"Learning is constructing or modifying representations of what is being experienced." --Ryszard Michalski

"Learning is making useful changes in our minds." --Marvin Minsky

What is machine learning ?

Generally speaking, the goal of machine learning is to build computer systems that can learn from their experience and adapt to their environments.

Why do we want machine learning?

If the program gets something right once it will always get it right. If it makes a mistake once it will always make the same mistake every time it runs. In contrast to computers, humans learn from their mistakes; attempt to work out why things went wrong and try alternative solutions. So, machine learning is a prerequisite for any mature programme of artificial intelligence.

What are the three phases in machine learning?

Machine learning typically follows three phases according to[Janet Finlay

1. Training: a training set of examples of correct behavior is analyzed and some representation of the newly learnt knowledge is stored. This is often some form of rules.

2. Validation: the rules are checked and, if necessary, additional training is given. Sometimes additional test data are used, but instead of using a human to validate the rules, some other automatic knowledge based component may be used. The role of tester is often called the critic.

3. Application: the rules are used in responding to some new situations

If example satisfies condition

Then assign it to class X

This sort of job classification is often termed as **concept learning.** The simplest case is when there are only two classes, of which one is seen as the desired "concept" to be learnt and the other is everything else. The "then" part of the rules is always the same and so the learnt rule is just a predicate describing the concept.

The difference between deductive reasoning and inductive reasoning is often used as the primary distinction between machine learning algorithms. Deductive learning working on existing facts and knowledge and deduces new knowledge from the old. In contrast, inductive learning uses examples and generates hypothesis based on the similarities between them.

Learning techniques available

Rote learning

In this kind of learning there is no prior knowledge. When a computer stores a piece of data, it is performing an elementary form of learning. This act of storage presumably allows the program to perform better in the future. Examples of correct behavior are stored and when a new situation arises it is matched with the learnt examples.

Deductive learning

Deductive learning works on existing facts and knowledge and deduces new knowledge from the old. This is best illustrated by giving an example. For example, assume:

A = B

B = C

Then we can deduce with much confidence that:

C = A

Arguably, deductive learning does not generate "new" knowledge at all, it simply memorizes the logical consequences of what is known already. This implies that virtually all mathematical research would not be classified as learning "new" things.

Inductive learning

Inductive learning takes examples and generalizes rather than starting with existing knowledge. For example, having seen many cats, all of which have tails, one might conclude that all cats have tails. There is scope of error in inductive reasoning, but still it is a useful technique that has been used as the basis of several successful systems. One major subclass of inductive learning is concept learning. This takes examples of a concept and tries to build a general description of the concept. The two very common inductive learning algorithms are version spaces and ID3.

How is it different from the AI we've studied so far?

Many practical applications of AI do not make use of machine learning. The relevant knowledge is built in at the start. Such programs even though are fundamentally limited; they are useful and do their job. However, even where we do not require a system to learn "on the job", machine learning has a part to play.

Machine learning in developing expert systems?

Many AI applications are built with rich domain knowledge and hence do not make use of machine learning. However, the fundamental problem remains unresolved, in the sense that things that are normally implicit inside the expert's head must be made explicit. This is not always easy as the experts may find it hard to say what rules they use to assess a situation but they can always tell you what factors they take into account. This is where machine learning mechanism could help. A machine learning

program can take descriptions of situations couched in terms of these factors and then infer rules that match expert's behavior.

Some of the applications that use learning algorithms include:

- Spoken digits and word recognition
- Handwriting recognition
- Driving autonomous vehicles
- Path finders
- Intelligent homes
- Intrusion detectors
- Intelligent refrigerators, tvs, vacuum cleaners
- Computer games
- Humanoid robotics

A general model of learning agents, pattern recognition

Any given learning problem is primarily composed of three things:

- Input
- Processing unit
- Output

The processing unit is the learning agent in our focus of study. Any learning agent or algorithm should in turn have at least the following **three characteristics:**

Feature representation

The input is usually broken down into a number of features. This is not a rule, but sometimes the real world problems have inputs that cannot be fed to a learning system directly, In reality, we usually associate some attributes or features to every input, for instance, two features that can define a student can be: grade and class participation. So these become the feature set of the learning system. Based on these features, the learner processes each input.

Distance measure

Given two different inputs, the learner should be able to tell them apart. The distance measure is the procedure that the learner uses to calculate the difference between the two inputs.

Generalization

In the training phase, the learner is presented with some positive and negative examples from which it leans. In the testing phase, when the learner comes across new but similar inputs, it should be able to classify them similarly. This is called generalization.

LEARNING: Symbol-based

the techniques we are to learn now use symbols to represent knowledge and information.

Problem and problem spaces

In theoretical computer science there are two main branches of problems:

- Tractable
- Intractable

Those problems that can be solved in polynomial time are termed as tractable, the other half is called intractable.

The tractable problems are further divided into structured and complex problems.
Structured problems are those which have defined steps through which the solution to the problem is reached. Complex problems usually don't have well-defined steps.
Machine learning algorithms are particularly more useful in solving the complex problems like recognition of patterns in images or speech, for which it's hard to come up with procedural algorithms otherwise.
The solution to any problem is a function that converts its inputs to corresponding outputs.

Instance space

How many distinct instances can the concept sick have? Since there are two attributes: T and BP, each having 3 values, there can be a total of 9 possible
distinct instances in all.

Concept space

A concept is the representation of the problem with respect to the given attributes, for example, if we're talking about the problem scenario of concept SICK defined over the attributes T and BP, then the concept space is defined by all the combinations of values of SK for every instance x.

Hypothesis space

just imagine how huge the concept space would be for real world problems that involve larger attribute sets. So the learner has to apply some hypothesis, which has either a search or the language bias to reduce the size of the concept space. This reduced concept space becomes the hypothesis space.

Version space and searching

Version space is a set of all the hypotheses that are consistent with all the training examples. When we are given a set of training examples D, it is possible that there might be more than one hypotheses from the hypothesis space that are consistent with all the training examples. By consistent we mean $h(x_i) = C(x_i)$. That is, if the true output of a concept $[c(x_i)]$ is 1 or 0 for an instance, then the output by our hypothesis $[h(x_i)]$ is 1 or 0 as well, respectively. If this is true for every instance in our training set D, we can say that the hypothesis is consistent

Concept learning as search

Now that we are well familiar with most of the terminologies of machine learning, we can define the learning process in technical terms as:
"We have to assume that the concept lies in the hypothesis space. So we search for a hypothesis belonging to this hypothesis space that best fits the training examples, such that the output given by the hypothesis is same as the true output of concept." The stress here is on the word 'search'. We need to somehow search through the hypothesis space.

General to specific ordering of hypothesis space

So all the hypothesis in H can be ordered according to their generality, starting from < ?, ? > which is the most general hypothesis since it always classifies all the instances as positive. On the contrary we have < Ø , Ø > which is the most specific hypothesis, since it doesn't classify a single instance as positive.

FIND-S

FIND-S finds the maximally specific hypothesis possible within the version space given a set of training data. How can we use the general to specific ordering of hypothesis space to organize the

search for a hypothesis consistent with the observed training examples? One way is to begin with the most specific possible hypothesis in H, then generalize the hypothesis each time it fails to cover an observed positive training example. (We say that a hypothesis "covers" a positive example if it correctly classifies the example as positive.) To be more precise about how the partial ordering is used, consider the FIND-S algorithm: There might be other hypotheses in the version space but this one was the maximally specific with respect to the given three training examples. For generalization purposes we might be interested in the other hypotheses but FIND-S fails to find the other hypotheses. Also in real world problems, the training data isn't consistent and void of conflicting errors. This is another **drawback of FIND-S**, that, it assumes the consistency within the training set.

Candidate-Elimination algorithm

The key idea in Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples. This subset of all hypotheses is actually the version space with respect to the hypothesis space H and the training examples D, because it contains all possible versions of the target concept.

The Candidate-Elimination algorithm represents the version space by storing only its most general members (denoted by G) and its most specific members (denoted by S). Given only these two sets S and G, it is possible to enumerate all members of the version space as needed by generating the hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses.

DECISION TREES LEARNING

Up until now we have been searching in conjunctive spaces which are formed by ANDing the attributes, for instance. But this is a very restrictive search, as we saw the reduction in hypothesis space from 2^9 total possible concepts to 17. This can be risky if we're not sure if the true concept will lie in the conjunctive space. So a safer approach is to relax the searching constraints. One way is to involve OR into the search. Do you think we'll have a bigger search space if we employ OR? Yes, most certainly. If we could use these kind of OR statements, we'd have a better chance of finding the true concept, if the concept does not lie in the conjunctive space. These are also called **disjunctive spaces**.

Decision tree representation

Decision trees give us disjunctions of conjunctions, that is, they have the form:

(A AND B) OR (C AND D)

where A, B, C and D are the attributes for the problem. This tree gives a positive output if either A AND B attributes are present in the instance; OR C AND D attributes are present. Through decision trees, this is how we reach the final hypothesis. This is a hypothetical tree. In real problems, every tree has to have a root node. There are various algorithms like ID3 and C4.5 to find decision trees for learning problems.

ID3

ID stands for interactive dichotomizer. This was the 3rd revision of the algorithm which got wide acclaims. The first step of ID3 is to find the root node. It uses a special function GAIN, to evaluate the gain information of each attribute. For example if there are 3 instances, it will calculate the gain

information for each. Whichever attribute has the maximum gain information, becomes the root node. The rest of the attributes then fight for the next slots.

Entropy

In order to define information gain precisely, we begin by defining a measure commonly used in statistics and information theory, called entropy, which characterizes the purity/impurity of an arbitrary collection of examples. Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is:

Entropy(S) = - $p_+ \log_2 p_+$ - $p_- \log_2 p$ where p_+ is the proportion of positive examples in S and p_- is the proportion of negative examples in S. In all calculations involving entropy we define 0log 0 to be 0.

Notice that the entropy is 0, if all the members of S belong to the same class (purity). For example, if all the members are positive. the entropy is 1 when the collection contains equal number of positive and negative examples (impurity). Otherwise if the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Information gain

we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called information gain, is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. That is, if we use the attribute with the maximum information gain as the node, then it will classify some of the instances as positive or negative with 100% accuracy, and this will reduce the entropy for the remaining instances.

LEARNING: CONNECTIONIST

ANNs can compute more complicated functions ranging from linear to any higher order quadratic, especially for non-Boolean concepts. This new learning paradigm takes its roots from biology inspired approach to learning.

Tasks for which connectionist approach is well suited include:

- Classification
- Fruits – Apple or orange
- Pattern Recognition
- Finger print, Face recognition
- Prediction
- Stock market analysis, weather forecast

Biological aspects and structure of a neuron

The brain is a collection of about 100 billion interconnected neurons. Each neuron is a cell that uses biochemical reactions to receive, process and transmit information. A neuron's dendritic tree is connected to a thousand neighboring neurons. When one of those neurons fire, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation.

Single perceptron

An artificial neuron is defined as follows:

It receives a number of inputs (either from original data, or from the output of other neurons in the neural network). Each input comes via a connection that has a strength (or weight); these weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value.

The weighted sum of the inputs is formed, and the threshold subtracted, to compose the activation of the neuron. The activation signal is passed through an activation function (also known as a transfer function) to produce the output of the neuron.

Response of changing bias

The response of changing the bias of a neuron results in shifting the decision line up or down. Response of changing weight The change in weight results in the rotation of the decision line. Hence this up and down shift, together with the rotation of the straight line can achieve any linear decision.

Linearly separable problems

There is a whole class of problems which are termed as linearly separable. This name is given to them, because if we were to represent them in the input space, we could classify them using a straight line. The simplest examples are the logical AND or OR.

A single perceptron simply draws a line, which is a hyper plane when the data is more then 2 dimensional. Sometimes there are complex problems (as is the case in real life). The data for these problems cannot be separated into their respective classes by using a single straight line. These problems are not linearly separable. Another example of linearly non-separable problems is the XOR gate (exclusive OR).

A single layer perceptron can perform pattern classification only on linearly separable patterns, regardless of the type of non-linearity. However, multi-layer perceptron and the back-propagation algorithm overcomes many of the shortcomings of the single layer perceptron.

Multiple layers of perceptrons

there are many problems which need to have multiple decision lines for a good acceptable solution. Multiple layer perceptrons achieve this task by the introduction of one or more hidden layers. Each neuron in the hidden layer is responsible for a different line. Together they form a classification for the given problem. Each neuron in the hidden layer forms a different decision line. Together all the lines can construct any arbitrary non-linear decision boundaries. These multilayer perceptrons are the most basic artificial neural networks.

ARTIFICIAL NEURAL NETWORKS: SUPERVISED AND UNSUPERVISED

A neural network is a massively parallel distributed computing system that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in **two respects**:

- Knowledge is acquired by the network through a learning process (called training)
- Interneuron connection strengths known as synaptic weights are used to store the knowledge

Knowledge in the artificial neural networks is implicit and distributed.

Advantages

- Excellent for pattern recognition
- Excellent classifiers
- Handles noisy data well
- Good for generalization

Draw backs

- The power of ANNs lie in their parallel architecture

– Unfortunately, most machines we have are serial (Von Neumann architecture)

• Lack of defined rules to build a neural network for a specific problem
– Too many variables, for instance, the learning algorithm, number of neurons per layer, number of layers, data representation etc
• Knowledge is implicit
• Data dependency

Basic terminologies

Number of layers
- o Single layer network
- o Multilayer networks

Single Layer
Only one input and
One output layer
Two Layers
One input ,
One hidden and
One output layer

Direction of information (signal) flow
o Feed-forward
o Recurrent (feed-back)

Connectivity
o Fully connected
o Partially connected

Learning methodology
o Supervised
Given a set of example input/output pairs, find a rule that does a good job of predicting the output associated with a new input.
o Unsupervised
Given a set of examples with no labeling, group them into sets called clusters

Knowledge is not explicitly represented in ANNs. Knowledge is primarily encoded in the weights of the neurons within the network

Design phases of ANNs

Feature Representation
• The number of features are determined using no of inputs for the problem.
• Text Classification (# of words)
• Gene Arrays for DNA classification (5,000-50,000)
- • Images (512 x 512)

• These large feature spaces make algorithms run slower. They also make the training process longer. The solution lies in finding a smaller feature space which is the subset of existing features.

• Feature Space should show discrimination between classes of the data.

Training

• Training is either supervised or unsupervised.

• Remember when we said:

• We assume that the concept lies in the hypothesis space. So we search for a hypothesis belonging to this hypothesis space that best fits the training examples.

• Finding the right hypothesis is the goal of the training session. So neural networks are doing function approximation, and training stops when it has found the closest possible function that gives the minimum error on all the instances

• Training is the heart of learning, in which finding the best hypothesis that covers most of the examples is the objective.

Learning is simply done through adjusting the weights of the Network

Similarity Measurement

• A measure to tell the difference between the actual output of the network while training and the desired labeled output

• The most common technique for measuring the total error in each iteration of the neural network (epoch) is Mean Squared Error (MSE).

Validation

• During training, training data is divided into k data sets; k-1 sets are used for training, and the remaining data set is used for cross validation. This ensures better results, and avoids over-fitting.

Stopping Criteria

• Done through MSE. We define a low threshold usually 0.01, which if reached stops the training data.

• Another stopping criterion is the number of epochs, which defines how many maximum times the data can be presented to the network for learning.

Application Testing

• A network is said to generalize well when the input-output relationship computed by the network is correct (or nearly so) for input-output pattern (test data) never used in creating and training the network.

Supervised

Given a set of example input/output pairs, find a rule that does a good job of predicting the output associated with a new input.

Back propagation algorithm

1. Randomize the weights {ws} to small random values (both positive and negative)
2. Select a training instance t, i.e.,
3. Apply the network input vector to network input
4. Calculate the network output vector {zk(t)}, k = 1,...,Nout
5. Calculate the errors for each of the outputs k , k=1,...,
6. Calculate the necessary updates for weights -ws in a way that minimizes this error
7. Adjust the weights of the network by - ws
8. Repeat steps for each instance (pair of input–output vectors) in the training set until the error for the entire system

Unsupervised

Given a set of examples with no labeling, group them into sets called clusters

- A cluster represents some specific underlying patterns in the data
- Useful for finding patterns in large data sets
- Form clusters of input data
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output

Self-organizing neural networks: clustering, quantization, function approximation, Kohonen maps

1. Each node's weights are initialized
2. A data input from training data (vector) is chosen at random and presented to the cluster lattice
3. Every cluster centre is examined to calculate which weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU)
4. The radius of the neighborhood of the BMU is now calculated. Any nodes found within this radius are deemed to be inside the BMU's neighborhood
5. Each neighboring node's (the nodes found in step 4) weights are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered
6. Repeat steps for N iterations

PLANNING

Definition of Planning

The field of acting logically to solve problems is known as Planning. Planning is based on logic representation that we have already studied. The key in planning is to use logic in order to solve problem elegantly. Planning is an advanced form problem solving which generates a sequence of operators that guarantee the goal. Furthermore, such sequence of operators or actions (commonly used in planning literature) is called a plan.

Planning vs. problem solving

Planning introduces the following improvements with respect to classical problem solving:

- Each state is represented in predicate logic. De-facto representation of a state is the conjunction (AND) of predicates that are true in that state.
- The goal is also represented as states, i.e. conjunction of predicates.
- Each action (or operator) is associated with some logic preconditions that must be true for that action to be applied. Thus a planning system can avoid any action that is just not possible at a particular state.
- Each action is associated with an 'effect' or post-conditions. These postconditions specify the added and/or deleted predicates when the action is applied.
- The inference mechanism used is that of backward chaining so as to use only the actions and states that are really required to reach goal state.
- Optional: The sequence of actions (plan) is minimally ordered. Only those actions are ordered in a sequence when any other order will not achieve the desired goal. Therefore, planning allows partial ordering.

Planning language

STRIPS is one of the founding languages developed particularly for planning.

Condition predicates

Condition predicates are the predicates that define states. For example, a predicate that specifies that we are at location 'X' is given as.

at(X)

State

State is a conjunction of predicates represented in well-known form, for example, a state where we are at the hotel and do not have either cash or radio is represented as,

$$at(hotel) \wedge \neg have(cash) \wedge \neg have(radio)$$

Goal

Goal is also represented in the same manner as a state. For example, if the goal of a planning problem is to be at the hotel with radio, it is represented as,

$$at(hotel) \wedge have(radio)$$

Action Predicates

Action is a predicate used to change states. It has three components namely, the predicate itself, the pre-condition, and post-condition predicates. For example, the action to buy something item can be represented as,

Action:

buy(X)

Pre-conditions:

at(Place) _ sells(Place, X)

Post-conditions/Effect:

have(X)

The partial-order planning algorithm – POP

You just start with an empty plan in which naturally, no condition predicate of goal state is met i.e. pre-conditions of finish action are not met. You backtrack by adding actions that meet these unsatisfied pre-condition predicates. New unsatisfied preconditions will be generated for each newly added action. Then you try to satisfy those by using appropriate actions in the same way as was done for goal state initially. You keep on doing that until there is no unsatisfied precondition. at some time there might be two actions at the same level of ordering of them one action's effect conflicts with other action's pre-condition. This is called a threat and should be resolved. Threats are resolved by simply reordering such actions such that you see no threat. Because this algorithm does not order actions unless absolutely necessary it is known as a partial-order planning algorithm.

ADVANCED TOPICS

Computer vision

It is a subfield of Artificial Intelligence. The purpose of computer vision is to study algorithms, techniques and applications that help us make machines that can "understand" images and videos. In other words, it deals with procedures that extract useful information from static pictures and sequence of images. Enabling a machine to see, percieve and understand exactly as humans see, percieve and understand is the aim of Computer Vision.

Computer vision finds its applications in medicine, military, security and surveillance, quality inspection, robotics, automotive industry and many other areas. Few areas of vision in which research is benig actively conducted throughout the world are as follows:

- The detection, segmentation, localisation, and recognition of certain objects in images (e.g., human faces)
- Tracking an object through an image sequence
- Object Extraction from a video sequence
- Automated Navigation of a robot or a vehicle
- Estimation of the three-dimensional pose of humans and their limbs
- Medical Imaging, automated analysis of different body scans (CT Scan, Bone Scan, X-Rays)
- Searching for digital images by their content (content-based image retrieval)
- Registration of different views of the same scene or object

Robotics

Robotics is the highly advanced and totally hyped field of today. Literally speaking, robotics is the study of robots. Robots are nothing but a complex combination of hardware and intelligence, or mechanics and brains. Thus robotics is truly a multi-disciplinary area, having active contributions from, physics, mechanics, biology, mathematics, computer science, statistics, control thory, philosophy, etc.

The features that constitute a robot are:

- Mobility
- Perception
- Planning
- Searching
- Reasoning
- Dealing with uncertainty
- Vision
- Learning
- Autonomy
- Physical Intelligence

Softcomputing

Softcomputing is a relatively new term coined to encapsulate the emergence of new hybrid area of work in AI. Different technologies including fuzzy systems, genetic algorithms, neural networks and a few statistical methods have been combined together in different orientations to successfully solve today's complex real-world problems.

The most common combinations are of the pairs

- genetic algorithms – fuzzy systems (genetic fuzzy)
- Neural Networks – fuzzy systems (neuro-fuzzy systems)
- Genetic algorithms – Neural Networks (neuro-genetic systems)

Clustering

Clustering is a form of unsupervised learning, in which the training data is available but without the classification information or class labels. The task of clustering is to identify and group similar

individual data elements based on some measure of similarity. So basically using clustering algorithms, classification information can be 'produced' from a training data which has no classification data at the first place. Naturally, there is no supervision of classification in clustering algorithms for their learning/clustering, and hence they fall under the category of unsupervised learning.
The famous clustering algorithms are Self-organizing maps (SOM), k-means, linear vector quantization, Density based data analysis, etc.