UNIT-05

Learning, Common Sense, Expert Systems

Introduction to Learning

Machine Learning is the study of how to build computer systems that adapt and improve with experience. It is a subfield of Artificial Intelligence and intersects with cognitive science, information theory, and probability theory, among others.

Classical AI deals mainly with *deductive* reasoning, learning represents *inductive* reasoning. Deductive reasoning arrives at answers to queries relating to a particular situation starting from a set of general axioms, whereas inductive reasoning arrives at general axioms from a set of particular instances.

Classical AI often suffers from the knowledge acquisition problem in real life applications where obtaining and updating the knowledge base is costly and prone to errors. Machine learning serves to solve the knowledge acquisition bottleneck by obtaining the result from data by induction.

Machine learning is particularly attractive in several real life problem because of the following reasons:

- Some tasks cannot be defined well except by example
- Working environment of machines may not be known at design time
- Explicit knowledge encoding may be difficult and not available
- Environments change over time
- Biological systems learn

Recently, learning is widely used in a number of application areas including,

- Data mining and knowledge discovery
- Speech/image/video (pattern) recognition
- Adaptive control
- Autonomous vehicles/robots
- Decision support systems
- Bioinformatics
- WWW

Formally, a computer program is said to **learn** from experience ***E*** with respect to some class of tasks ***T*** and performance measure ***P***, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

Thus a learning system is characterized by:

A task T

B experience E, and

C performance measure P

Examples:*Learning to play chess*

T: Play chess

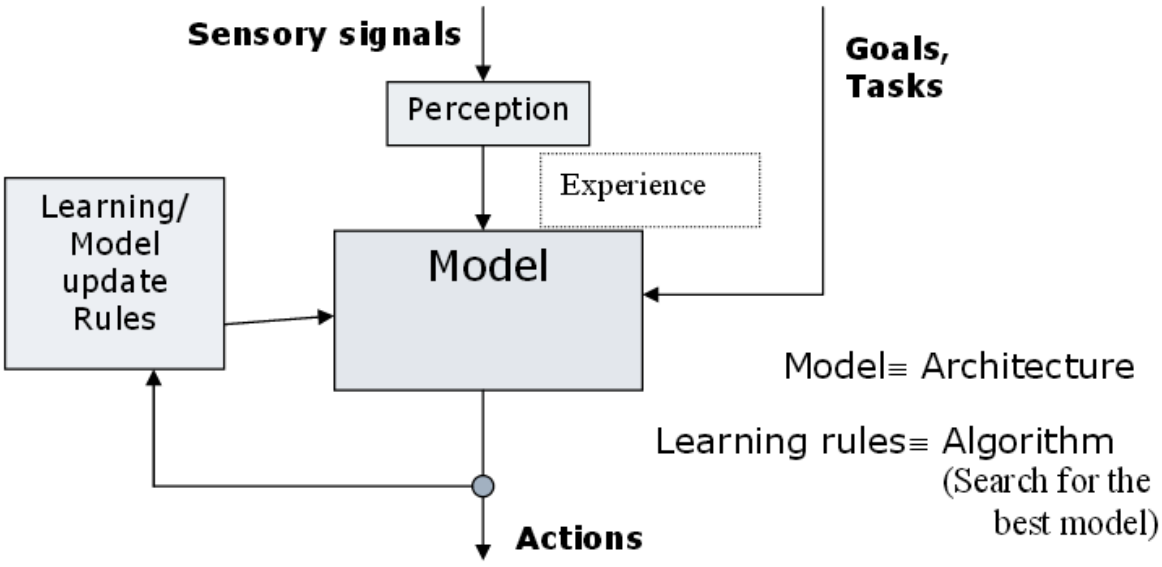
P: Percentage of games won in world tournament
E: Opportunity to play against self or other players

Learning to drive a van

T: Drive on a public highway using vision sensors

P: Average distance traveled before an error (according to human observer)
E: Sequence of images and steering actions recorded during human driving.

The block diagram of a generic learning system which can realize the above definition is shown below:



\

As can be seen from the above diagram the system consists of the following components:

1. *Goal:* Defined with respect to the task to be performed by the system
2. *Model:* A mathematical function which maps perception to actions

3. *Learning rules:* Which update the model parameters with new experience such that the performance measures with respect to the goals is optimized
4. *Experience:* A set of perception (and possibly the corresponding actions)

12.1.1 Taxonomy of Learning Systems

Several classification of learning systems are possible based on the above components as follows:

Goal/Task/Target Function:

Prediction: To predict the desired output for a given input based on previous input/output pairs. E.g., to predict the value of a stock given other inputs like market index, interest rates etc.

Categorization: To classify an object into one of several categories based on features of the object. E.g., a robotic vision system to categorize a machine part into one of the categories, spanner, hammer etc based on the parts' dimension and shape.

Clustering: To organize a group of objects into homogeneous segments. E.g., a satellite image analysis system which groups land areas into forest, urban and water body, for better utilization of natural resources.

Planning: To generate an optimal sequence of actions to solve a particular problem. E.g., an Unmanned Air Vehicle which plans its path to obtain a set of pictures and avoid enemy anti-aircraft guns.

Models:

- Propositional and FOL rules
- Decision trees
- Linear separators
- Neural networks
- Graphical models
- Temporal models like hidden Markov models

Learning Rules:

Learning rules are often tied up with the model of learning used. Some common rules are gradient descent, least square error, expectation maximization and margin maximization.

Experiences:

Learning algorithms use experiences in the form of perceptions or perception action pairs to improve their performance. The nature of experiences available varies with applications. Some common situations are described below.

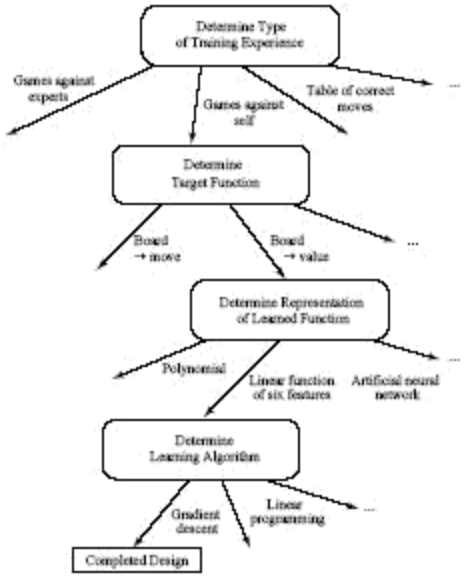
Supervised learning: In supervised learning a teacher or oracle is available which provides the desired action corresponding to a perception. A set of perception action pair provides what is called a training set. Examples include an automated vehicle where a set of vision inputs and the corresponding steering actions are available to the learner.

Unsupervised learning: In unsupervised learning no teacher is available. The learner only discovers persistent patterns in the data consisting of a collection of perceptions. This is also called exploratory learning. Finding out malicious network attacks from a sequence of anomalous data packets is an example of unsupervised learning.

Active learning: Here not only a teacher is available, the learner has the freedom to ask the teacher for suitable perception-action example pairs which will help the learner to improve its performance. Consider a news recommender system which tries to learn an users preferences and categorize news articles as interesting or uninteresting to the user. The system may present a particular article (of which it is not sure) to the user and ask whether it is interesting or not.

Reinforcement learning: In reinforcement learning a teacher is available, but the teacher instead of directly providing the desired action corresponding to a perception, return reward and punishment to the learner for its action corresponding to a perception. Examples include a robot in a unknown terrain where its get a punishment when its hits an obstacle and reward when it moves smoothly.

In order to design a learning system the designer has to make the following choices based on the application.



12.1.2 Mathematical formulation of the inductive learning problem

- Extrapolate from a given set of examples so that we can make accurate predictions about future examples.
- **Supervised versus Unsupervised learning**
  Want to learn an unknown function $f(\mathbf{x}) = \mathbf{y}$, where **x** is an input example and **y** is the desired output. Supervised learning implies we are given a set of (**x, y**) pairs by a "teacher." Unsupervised learning means we are only given the **x**s. In either case, the goal is to estimate *f*.

Inductive Bias

- Inductive learning is an inherently conjectural process because any knowledge created by generalization from specific facts cannot be proven true; it can only be proven false. Hence, inductive inference is **falsity preserving**, not truth preserving.
- To generalize beyond the specific training examples, we need constraints or **biases** on what *f* is best. That is, learning can be viewed as searching the **Hypothesis Space** H of possible *f* functions.
- A bias allows us to choose one *f* over another one
- A completely unbiased inductive algorithm could only memorize the training examples and could not say anything more about other unseen examples.
- Two types of biases are commonly used in machine learning:
  - **Restricted Hypothesis Space Bias**
    Allow only certain types of *f* functions, not arbitrary ones
  - **Preference Bias**

Define a metric for comparing *f*s so as to determine whether one is better than another

Inductive Learning Framework

- Raw input data from sensors are preprocessed to obtain a **feature vector**, **x**, that adequately describes all of the relevant features for classifying examples.
- Each **x** is a list of (attribute, value) pairs. For example,

  x = `(Person = Sue, Eye-Color = Brown, Age = Young, Sex = Female)`

  The number of attributes (also called features) is fixed (positive, finite). Each attribute has a fixed, finite number of possible values.

Each example can be interpreted as a *point* in an *n*-dimensional **feature space**, where *n* is the number of attributes.

Neural Networks

Artificial neural networks are among the most powerful learning models. They have the versatility to approximate a wide range of complex functions representing multi-dimensional input-output maps. Neural networks also have inherent adaptability, and can perform robustly even in noisy environments.

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected simple processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. ANNs can process information at a great speed owing to their highly massive parallelism.

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Other advantages include:

- Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
- Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
- Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

Artificial Neural Networks

Artificial neural networks are represented by a set of nodes, often arranged in layers, and a set of weighted directed links connecting them. The nodes are equivalent to neurons, while the links denote synapses. The nodes are the information processing units and the links acts as communicating media.

There are a wide variety of networks depending on the nature of information processing carried out at individual nodes, the topology of the links, and the algorithm for adaptation of link

weights. Some of the popular among them include:

Perceptron: This consists of a single neuron with multiple inputs and a single output. It has restricted information processing capability. The information processing is done through a transfer function which is either linear or non-linear.

Multi-layered Perceptron (MLP): It has a layered architecture consisting of input, hidden and output layers. Each layer consists of a number of perceptrons. The output of each layer is transmitted to the input of nodes in other layers through weighted links. Usually, this transmission is done only to nodes of the next layer, leading to what are known as feed forward networks. MLPs were proposed to extend the limited information processing capabilities of simple percptrons, and are highly versatile in terms of their approximation ability. Training or weight adaptation is done in MLPs using supervised backpropagation learning.

Recurrent Neural Networks: RNN topology involves backward links from output to the input and hidden layers. The notion of time is encoded in the RNN information processing scheme. They are thus used in applications like speech processing where inputs are time sequences data.

Self-Organizing Maps: SOMs or Kohonen networks have a grid topology, wit unequal grid weights. The topology of the grid provides a low dimensional visualization of the data distribution. These are thus used in applications which typically involve organization and human browsing of a large volume of data. Learning is performed using a winner take all strategy in a unsupervised mode.

Common Sense

In artificial intelligence research, commonsense knowledge is the collection of facts and information that an ordinary person is expected to know. The commonsense knowledge problem is the ongoing project in the field of knowledge representation (a sub-field of artificial intelligence) to create a commonsense knowledge base: a database containing all the general knowledge that most people possess, represented in a way that it is available to artificial intelligence programs that use natural language or make inferences about the ordinary world. Such a database is a type of ontology of which the most general are called upper ontologies.

The problem is considered to be among the hardest in all of AI research because the breadth and detail of commonsense knowledge is enormous. Any task that requires commonsense knowledge is considered AI-complete: to be done as well as a human being does it, it requires the machine to appear as intelligent as a human being. These tasks include machine translation, object recognition, text mining and many others. To do these tasks perfectly, the machine simply has to know what the text is talking about or what objects it may be looking at, and this is impossible in general unless the machine is familiar with all the same concepts that an ordinary person is familiar with.

Information in a commonsense knowledge base may include, but is not limited to, the following:

- An ontology of classes and individuals
- Parts and materials of objects
- Properties of objects (such as color and size)
- Functions and uses of objects
- Locations of objects and layouts of locations
- Locations of actions and events
- Durations of actions and events
- Preconditions of actions and events
- Effects (postconditions) of actions and events
- Subjects and objects of actions
- Behaviors of devices
- Stereotypical situations or scripts
- Human goals and needs
- Emotions
- Plans and strategies
- Story themes
- Contexts

Expert Systems

An **expert system** is a computer program that contains some of the subject-specific knowledge of one or more human experts. An expert systems are meant to solve real problems which normally would require a specialized human expert (such as a doctor or a minerologist). Building an expert system therefore first involves extracting the relevant knowledge from the human expert. Such knowledge is often heuristic in nature, based on useful ``rules of thumb'' rather than absolute certainties. Extracting it from the expert in a way that can be used by a computer is generally a difficult task, requiring its own expertise. A *knowledge engineer* has the job of extracting this knowledge and building the expert system *knowledge base*.

A first attempt at building an expert system is unlikely to be very successful. This is partly because the expert generally finds it very difficult to express exactly what knowledge and rules they use to solve a problem. Much of it is almost subconscious, or appears so obvious they don't even bother mentioning it. *Knowledge acquisition* for expert systems is a big area of research, with a wide variety of techniques developed. However, generally it is important to develop an initial prototype based on information extracted by interviewing the expert, then iteratively refine it based on feedback both from the expert and from potential users of the expert system.

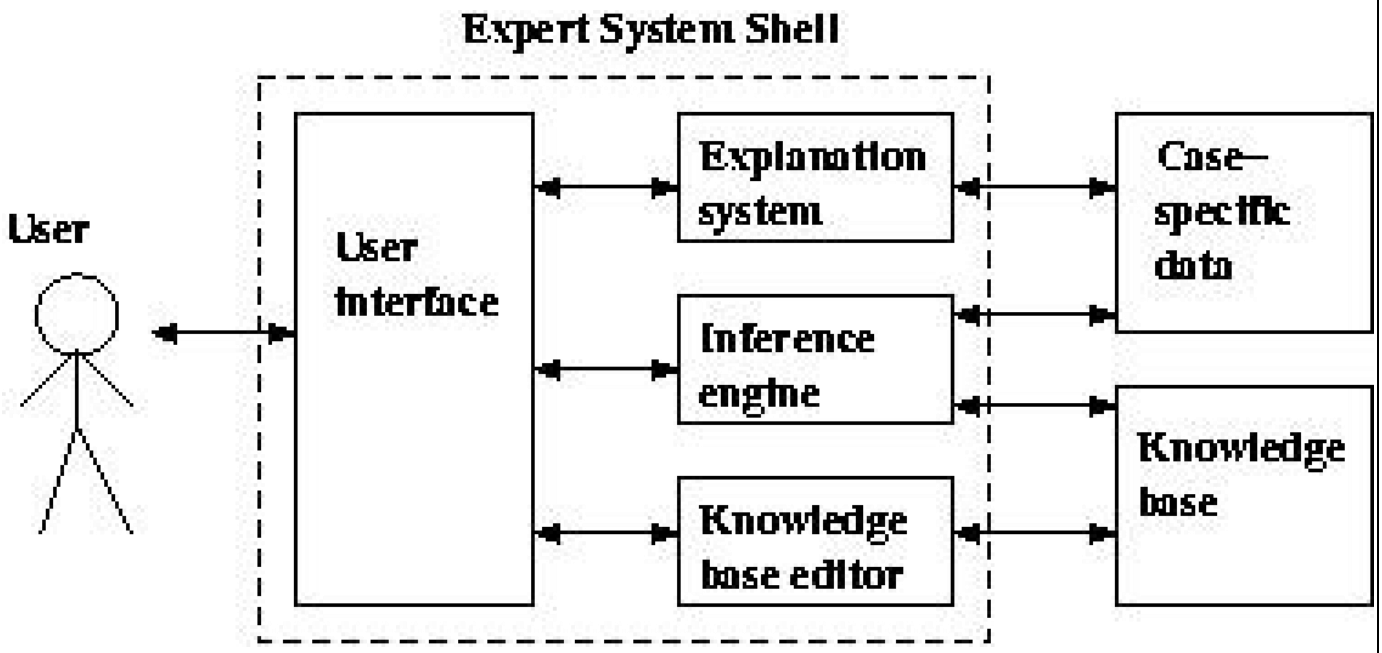
In order to do such iterative development from a prototype it is important that the expert system is written in a way that it can easily be inspected and modified. The system should be able to explain its reasoning (to expert, user and knowledge engineer) and answer questions about the solution process. Updating the system shouldn't involve rewriting a whole lot of code - just adding or deleting localized chunks of knowledge.

The most widely used knowledge representation scheme for expert systems is rules. Typically, the rules won't have certain conclusions - there will just be some degree of certainty that the conclusion will hold if the conditions hold. Statistical techniques are used to determine these certainties. Rule-based systems, with or without certainties, are generally easily modifiable and make it easy to provide reasonably helpful traces of the system's reasoning. These traces can be used in providing explanations of what it is doing.

Expert systems have been used to solve a wide range of problems in domains such as medicine, mathematics, engineering, geology, computer science, business, law, defence and education. Within each domain, they have been used to solve problems of different types. Types of problem involve *diagnosis* (e.g., of a system fault, disease or student error); *design* (of a computer systems, hotel etc); and *interpretation* (of, for example, geological data). The appropriate problem solving technique tends to depend more on the problem type than on the domain. Whole books have been written on how to choose your knowledge representation and reasoning methods given characteristics of your problem. https://www.rgpvonline.com

The following figure shows the most important modules that make up a rule-based expert system. The user interacts with the system through a *user interface* which may use menus, natural language or any other style of interaction). Then an *inference engine* is used to reason

with both the *expert knowledge* (extracted from our friendly expert) and data specific to the particular problem being solved. The expert knowledge will typically be in the form of a set of IF-THEN rules. The *case specific data* includes both data provided by the user and partial conclusions (along with certainty measures) based on this data. In a simple forward chaining rule-based system the case specific data will be the elements in *working memory*.



Almost all expert systems also have an *explanation subsystem*, which allows the program to explain its reasoning to the user. Some systems also have a *knowledge base editor* which help the expert or knowledge engineer to easily update and check the knowledge base.

One important feature of expert systems is the way they (usually) separate domain specific knowledge from more general purpose reasoning and representation techniques. The general purpose bit (in the dotted box in the figure) is referred to as an *expert system shell*. As we see in the figure, the shell will provide the inference engine (and knowledge representation scheme), a user interface, an explanation system and sometimes a knowledge base editor. Given a new kind of problem to solve (say, car design), we can usually find a shell that provides the right sort of support for that problem, so all we need to do is provide the expert knowledge. There are numerous commercial expert system shells, each one appropriate for a slightly different range of problems. (Expert systems work in industry includes both writing expert system shells and writing expert systems using shells.) Using shells to write expert systems generally greatly reduces the cost and time of development.