

TECHNICAL BENEFITS OF DIFFERENT TECHNIQUES IN ARTIFICIAL INTELLIGENCE

Introduction

Many activities involve intelligent action—problem solving, perception, learning, planning and other symbolic reasoning, creativity, language, and so forth—and therein lie an immense diversity of phenomena. Scientific concern for these phenomena is shared by many fields, for example, psychology, linguistics, and philosophy of mind, in addition to artificial intelligence. The starting point for artificial intelligence is the capability of the computer to manipulate symbolic expressions that can represent all manner of things, including knowledge about the structure and function of objects and people in the world, beliefs and purposes, scientific theories, and the programs of action of the computer itself.

Artificial intelligence is primarily concerned with symbolic representations of knowledge and heuristic methods of reasoning, that is, using common assumptions and rules of thumb. Two examples of problems studied in artificial intelligence are planning how a robot, or person, might assemble a complicated device, or move from one place to another; and diagnosing the nature of a person's disease, or of a machine's malfunction, from the observable manifestations of the problem. In both cases, reasoning with symbolic descriptions predominates over calculating.

The approach of artificial intelligence researchers is largely experimental, with small patches of mathematical theory. As in other experimental sciences, investigators build devices (in this case, computer programs) to carry out their experimental investigations. New programs are created to explore ideas about how intelligent action might be attained, and are also developed to test hypotheses about concepts or mechanisms involved in intelligent behavior.

The foundations of artificial intelligence are divided into representation, problem-solving methods, architecture, and knowledge. To work on a task, a computer must have an internal representation in its memory, for example, the symbolic description of a room for a moving robot, or a set of features describing a person with a disease. The representation also includes all the knowledge, including basic programs, for testing and measuring the structure, plus all the programs for transforming the structure into another one in ways appropriate to the task. Changing the representation used for a task can make an immense difference, turning a problem from impossible to trivial.

Given the representation of a task, a method must be adopted that has some chance of accomplishing the task. Artificial intelligence has gradually built up a stock of relevant problem-solving methods (the so-called weak methods) that apply extremely generally.

An important feature of all the weak methods is that they involve search. One of the most important generalizations to arise in artificial intelligence is the ubiquity of search. It appears to underlie all intelligent action. In the worst case, the search is blind. In heuristic search extra information is used to guide the search.

Some of the weak methods are generate-and-test (a sequence of candidates is generated, each being tested for solutionhood); hill climbing (a measure of progress is used to guide each step); means-ends analysis (the difference between the desired situation and the present one is used to select the next step); impasse resolution (the inability to take the desired next step leads to a subgoal of making the step feasible); planning by abstraction (the task is simplified, solved, and the solution used as a guide); and matching (the present situation is represented as a schema to be mapped into the desired situation by putting the two in correspondence).

An intelligent agent—person or program—has multiple means for representing tasks and dealing with them. Also required is an architecture or operating framework within which to select and carry out these activities. Often called the executive or control structure, it is best viewed as a total architecture (as in computer architecture), that is, a machine that provides data structures, operations on those data structures, memory for holding data structures, accessing operations for retrieving data structures from memory, a programming language for expressing integrated patterns of conditional operations, and an interpreter for carrying out programs. Any digital computer provides an architecture, as does any programming language. Architectures are not all equivalent, and one important scientific question is what architecture is appropriate for a general intelligent agent.

In artificial intelligence, the basic paradigm of intelligent action is that of search through a space of partial solutions (called the problem space) for a goal situation. Each step offers several possibilities, leading to a cascading of possibilities that can be represented as a branching tree. The search is thus said to be combinatorial or exponential. For example, if there are 10 possible actions in any situation, and it takes a sequence of 12 steps to find a solution (a goal state), then there are 10^{12} possible sequences in the exhaustive search tree. What keeps the search under control is knowledge, which suggests how to choose or narrow the options at each step. Thus the fourth fundamental concern is how to represent knowledge in the memory of the system so it can be brought to bear on the search when relevant.

An intelligent agent will have immense amounts of knowledge. This implies another major problem, that of discovering the relevant knowledge as the solution attempt progresses. Although this search does not include the combinatorial explosion characteristic of searching the problem space, it can be time consuming and hard. However, the structure of the database holding the knowledge (called the knowledge

base) can be carefully tailored to suit the architecture in order to make the search efficient. This knowledge base, with its accompanying problems of encoding and access, constitutes the final ingredient of an intelligent system.

An example of artificial intelligence is computer perception. Perception is the formation, from a sensory signal, of an internal representation suitable for intelligent processing. Though there are many types of sensory signals, computer perception has focused on vision and speech. Perception might seem to be distinct from intelligence, since it involves incident time-varying continuous energy distributions prior to interpretation in symbolic terms. However, all the same ingredients occur: representation, search, architecture, and knowledge. Speech perception starts with the acoustic wave of a human utterance and proceeds to an internal representation of what the speech is about. A sequence of representations is used: the digitization of the acoustic wave into an array of intensities; the formation of a small set of parametric quantities that vary continuously with time (such as the intensities and frequencies of the formants, bands of resonant energy characteristic of speech); a sequence of phons (members of a finite alphabet of labels for characteristic sounds, analogous to letters); a sequence of words; a parsed sequence of words reflecting grammatical structure; and finally a semantic data structure representing a sentence (or other utterance) that reflects the meaning behind the sounds.

A class of artificial intelligence programs called expert systems attempt to accomplish tasks by acquiring and incorporating the same knowledge that human experts have. Many attempts to apply artificial intelligence to medicine, government, and other socially significant tasks take the form of expert systems. Even though the emphasis is on knowledge, all the standard ingredients are present.

In careful tests, a number of expert systems have shown performance at levels of quality equivalent to or better than average practicing professionals (for example, average practicing physicians) on the restricted domains over which they operate. Nearly all large corporations and many smaller ones use expert systems. A common application is to provide technical assistance to persons who answer customers' trouble calls. Computer companies use expert systems to assist in configuring components from a parts catalog into a complete system that matches a customer's specifications, a kind of application that has been replicated in other industries tailoring assembled products to customers' needs. Troubleshooting and diagnostic programs are commonplace. Another widespread use of this technology is in software for home computers that assists taxpayers. One important lesson learned from incorporating artificial intelligence software into ongoing practice is that its success depends on many other aspects besides the intrinsic intellectual quality, for example, ease of interaction, integration into existing workflow, and costs.

Expert systems have sparked important insights in reasoning under uncertainty, causal reasoning, reasoning about knowledge, and acceptance of computer systems in the workplace. They illustrate that there is no hard separation between pure and applied artificial intelligence; finding what is required for intelligent action in a complex applied area makes a significant contribution to basic knowledge. *See also* Expert systems.

In addition to the subject areas mentioned above, significant work in artificial intelligence has been done on puzzles and reasoning tasks, induction and concept identification, symbolic mathematics, theorem proving in formal logic, natural language understanding and generation, vision, robotics, chemistry, biology, engineering analysis, computer-assisted instruction, and computer-program synthesis and verification, to name only the most prominent. As computers become smaller and less expensive, more and more intelligence is built into automobiles, appliances, and other machines, as well as computer software, in everyday use. *See also* Automata theory; Computer; Control systems; Cybernetics; Digital computer; Intelligent machine; Robotics.

The history of artificial intelligence (AI) predates the development of the first computing machines. On a general level, intelligence has been the subject of philosophical study for 2000 years. At the computational level, mathematician Alan Turing constructed a framework for AI during the era of analog computers.

While precise definitions are still the subject of debate, AI may be usefully thought of as *the branch of computer science that is concerned with the automation of intelligent behavior*. The intent of AI is to develop systems that have the ability to perceive and to learn, to accomplish physical tasks, and to emulate human decision making. AI seeks to design and develop intelligent agents as well as to understand them. Currently, the main fields of research and development include the following:

1. *Natural languages*: These studies focus on problems related to natural language interface, machine translation, understanding spoken language, and so forth.
2. *Expert systems*: No generalizable solutions are researched, but expertise is used to deal with ill-defined problems and relationships.
3. *Cognition and learning*: Investigations are being made into modes of thinking, learning, and problem solving.
4. *Computer vision*: Efforts are being made to develop principles and algorithms for machine vision and the interpretation of visual data.
5. *Automatic deduction*: This area deals with the resolution of problems, theorem proving, and logic programming.

Foundations

The term "AI" was applied about 1956, giving a formal name to work that had been developing over the previous five or six years. Individuals and organizations have an abiding interest in AI for several important reasons, including the following:

1. To preserve expertise that might be lost when an acknowledged expert is unavailable.
2. To create organizational knowledge bases so that others may learn from past problem-solving successes.
3. To help decision makers be consistent in their evaluation of complex problems.

During its early years AI was dominated by reliance on logic as a means of representing knowledge and on logical inference as the primary mechanism for intelligent reasoning. In the 1990s other paradigms arrived on the scene, some of which had a dramatic impact. *Artificial neural networks* (ANNs) were motivated by assumptions about how the brain functions— particularly the ideas of massively parallel connections, each of which performs simple computational tasks. Taken together, they represent knowledge as a property of patterns of relationships. *Genetic algorithms* apply principles of biological evolution to the problems of searching complex solution spaces. The programs do not use logical reasoning either, but evolve toward better and better solutions to complex problems.

Multiagent systems have recently come to the fore of AI research. This emergence has been driven by a recognition that intelligence may be reflected by the collective behaviors of large numbers of very simple interacting members of a community of agents. These agents can be computers, software modules, or virtually any object that can perceive aspects of its environment and proceed in a rational way toward accomplishing a goal.

A variety of disciplines have influenced the development of AI. These include philosophy (logic), mathematics (intractability, computability, algorithms), psychology (cognition), engineering (computer hardware and software), and linguistics (knowledge representation and natural-language processing).

Long before the development of computers, the notion that thinking was a form of computation motivated the formalization of logic. These efforts continue today. Graph theory provided the architecture for searching a solution space for a problem. Operations research, with its focus on optimization algorithms, used graph theory and other methods to solve complex decision-making problems.

In 1950, Alan Turing proposed what has become known as the Turing Test for defining intelligent behavior. The idea was to specify requirements that a computer would have to exhibit in order to demonstrate intelligence. Briefly, the Turing Test proposes that the computer should be interrogated via telecommunications by a human. Intelligence is exhibited by the computer if the interrogator cannot tell whether there is a human or a computer at the other end. In order to pass the test, a computer would need to have capabilities for natural-language processing, knowledge representation, automated reasoning, and machine learning.

An Evolution of Applications

While computer systems have become commonplace, they are generally rigid, complex, and incapable of rapid change. According to *A Report to ARPA on Twenty-First Century Intelligent Systems*, for us and our organizations to cope with the unpredictable eventualities of an ever-more volatile world, these systems need capabilities that will enable them to adapt readily to change. The report argues that our national

competitiveness depends increasingly on capacities for accessing, processing, and analyzing information (Grosz and Davis, 1994).

One of the early milestones in AI was Newell and Simon's General Problem Solver (GPS). The program was designed to imitate human problem-solving methods. This and other developments such as Logic Theorist and the Geometry Theorem Prover generated enthusiasm for the future of AI. Simon went so far as to assert that in the near-term future the problems that computers could solve would be coextensive with the range of problems to which the human mind has been applied.

Soon difficulties in achieving this objective began to manifest themselves. In scaling up from earlier successes, problems of intractability were encountered. A search for alternative approaches led to attempts to solve typically occurring cases in narrow areas of expertise. This prompted the development of expert systems. A seminal model was MYCIN, developed to diagnose blood infections. Having about 450 rules, MYCIN was able to perform as well as many experts. This and other expert-systems research led to the first commercial expert system, R1, implemented at Digital Equipment Corporation (DEC) to help configure orders for new computer systems. Sub-sequent to R1's implementation, it was estimated to save DEC about \$40 million a year.

Other classic systems include the PROSPECTOR program for determining the probable location and type of ore deposits and the INTERNIST program for performing medical diagnosis in internal medicine.

Branches of AI

Here's a list, but some branches are surely missing, because no-one has identified them yet. Some of these may be regarded as concepts or topics rather than full branches.

logical AI

What a program knows about the world in general the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals. The first article proposing this was [McC59]. [McC89] is a more recent summary. [McC96b] lists some of the concepts involved in logical ai. [Sha97] is an important text.

search

AI programs often examine large numbers of possibilities, e.g. moves in a chess game or inferences by a theorem proving program. Discoveries are continually made about how to do this more efficiently in various domains.

pattern recognition

When a program makes observations of some kind, it is often programmed to compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns, e.g. in a natural language text, in a chess position, or in the

history of some event are also studied. These more complex patterns require quite different methods than do the simple patterns that have been studied the most.

representation

Facts about the world have to be represented in some way. Usually languages of mathematical logic are used.

inference

From some facts, others can be inferred. Mathematical logical deduction is adequate for some purposes, but new methods of *non-monotonic* inference have been added to logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default, but the conclusion can be withdrawn if there is evidence to the contrary. For example, when we hear of a bird, we may infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Ordinary logical reasoning is monotonic in that the set of conclusions that can be drawn from a set of premises is a monotonic increasing function of the premises. Circumscription is another form of non-monotonic reasoning.

common sense knowledge and reasoning

This is the area in which AI is farthest from human-level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress, e.g. in developing systems of *non-monotonic reasoning* and theories of action, yet more new ideas are needed. The Cyc system contains a large but spotty collection of common sense facts.

learning from experience

Programs do that. The approaches to AI based on *connectionism* and *neural nets* specialize in that. There is also learning of laws expressed in logic. [Mit97] is a comprehensive undergraduate text on machine learning. Programs can only learn what facts or behaviors their formalisms can represent, and unfortunately learning systems are almost all based on very limited abilities to represent information.

planning

Planning programs start with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, they generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

epistemology

This is a study of the kinds of knowledge that are required for solving problems in the world.

ontology

Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are. Emphasis on ontology begins in the 1990s.

heuristics

A heuristic is a way of trying to discover something or an idea imbedded in a program. The term is used variously in AI. *Heuristic functions* are used in some

approaches to search to measure how far a node in a search tree seems to be from a goal. *Heuristic predicates* that compare two nodes in a search tree to see if one is better than the other, i.e. constitutes an advance toward the goal, may be more useful. [My opinion].

genetic programming

Genetic programming is a technique for getting programs to solve a task by mating random Lisp programs and selecting fittest in millions of generations. It is being developed by John Koza's group and here's a [tutorial](#).



Garry Kasparov playing against Deep Blue, the first machine to win a chess match against a reigning world champion.

Artificial intelligence (AI) is both the intelligence of machines and the branch of computer science which aims to create it.

Major AI textbooks define artificial intelligence as "the study and design of intelligent agents," where an intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success. AI can be seen as a realization of an abstract intelligent agent (AIA) which exhibits the functional essence of intelligence. John McCarthy, who coined the term in 1956, defines it as "the science and engineering of making intelligent machines."

Among the traits that researchers hope machines will exhibit are reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects. General intelligence (or "strong AI") has not yet been achieved and is a long-term goal of AI research.

AI research uses tools and insights from many fields, including computer science, psychology, philosophy, neuroscience, cognitive science, linguistics, ontology, operations research, economics, control theory, probability, optimization and logic. AI research also overlaps with tasks such as robotics, control systems, scheduling, data mining, logistics, speech recognition, facial recognition and many others. Other names for the field have been proposed, such as computational intelligence, synthetic intelligence, intelligent systems, or computational rationality.

Advantages of Artificial Intelligence in Virtual Worlds

While we already deal with some virtual AI -- notably in action games against computer-controlled "bots" or challenging a computer opponent to chess -- the work of Novamente, Electric Sheep Company and other firms has the potential to initiate a new age of virtual AI, one where, for better or worse, humans and artificial intelligences could potentially be indistinguishable.

If you think about it, we take in numerous pieces of information just walking down the street, much of it unconsciously. You might be thinking about the weather, the pace of your steps, where to step next, the movement of other people, smells, sounds, the distance to the destination, the effect of the environment around you and so forth. An artificial intelligence in a virtual world has fewer of these variables to deal with because as of yet, no virtual world approaches the complexity of the real world. It may be that by simplifying the world in which the artificial intelligence operates (and by working in a self-contained world), some breakthroughs can be achieved. Such a process would allow for a more linear development of artificial intelligence rather than an attempt to immediately jump to lifelike robots capable of learning, reason and self-analysis.

Goertzel states that a virtual world also offers the advantage of allowing a newly formed artificial intelligence to interact with thousands of people and characters, increasing learning opportunities [source: [PC World](#)]. The virtual body is also easier to manage and control than that of a robot. If an AI-controlled parrot seems to have particular challenges in a game world, it's less difficult for programmers to create another virtual animal than if they were working with a robot. And while a virtual world AI lacks a physical body, it displays more complexity (and more realism) than a simple AI that merely carries on text-based conversations with a human.

Novamente claims that its system is the first to allow artificial intelligences to progress through a process of self-analysis and learning [source: [Novamente](#)]. The company hopes that its AI will also distinguish itself from other attempts at AI by surprising its creators in its capabilities -- for example, by learning a skill or task that it wasn't programmed to perform. Novamente has already created what it terms an "artificial baby" in the AGISim virtual world [source: [Novamente](#)]. This artificial baby has learned to perform some basic functions.



© Photographer: Alice Dehaven~herden
Agency: [Dreamstime.com](#)
Virtual artificial intelligences are a long way from realistic androids, but they may help in the development of more advanced AI.

Despite all of this excitement, the AI discussed here are far from what's envisioned in "Terminator." It will be some time before AIs are seamlessly interacting with players, impressing us with their cleverness and autonomy and seeming all too human. Even Philip Rosedale, the founder of Linden Labs, the company behind "Second Life," has warned against becoming caught up in the hype of the supposedly groundbreaking potential of these virtual worlds [source: CNET News].

But "Second Life" and other virtual worlds may prove to be the most valuable testing grounds to date for AI. It will also be interesting to track how virtual artificial intelligences progress as the virtual worlds they occupy change and become more complex. Besides acting as an incubator for artificial intelligence, "Second Life" has already been an important case study in the development of cyber law and the economics and legality of hawking virtual goods for real dollars. The popular virtual world has even been mentioned as a possible virtual training facility for children taking emergency preparedness classes [source: CNET News].

NEURAL NETWORK

What is a neural network?

Neural Networks are a different paradigm for computing:

- von Neumann machines are based on the processing/memory abstraction of human information processing.
- neural networks are *based on the parallel architecture of animal brains*.

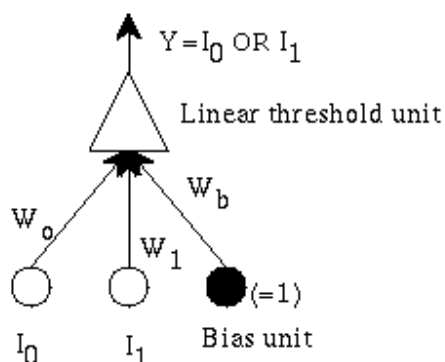
Neural networks are a form of multiprocessor computer system, with

- simple processing elements
- a high degree of interconnection
- simple scalar messages
- adaptive interaction between elements

A biological neuron may have as many as 10,000 different inputs, and may send its output (the presence or absence of a short-duration spike) to many other neurons. Neurons are wired up in a 3-dimensional pattern.

Real brains, however, are orders of magnitude more complex than any artificial neural network so far considered.

Example: A simple single unit adaptive network:



The network has 2 inputs, and one output. All are binary. The output is

$$1 \text{ if } W_0 * I_0 + W_1 * I_1 + W_b > 0$$

0 if $W_0 * I_0 + W_1 * I_1 + W_b \leq 0$

We want it to learn simple OR: output a 1 if either I_0 or I_1 is 1.

Algorithms and Architectures.

The simple Perceptron:

The network adapts as follows: *change the weight by an amount proportional to the difference between the desired output and the actual output.*

As an equation:

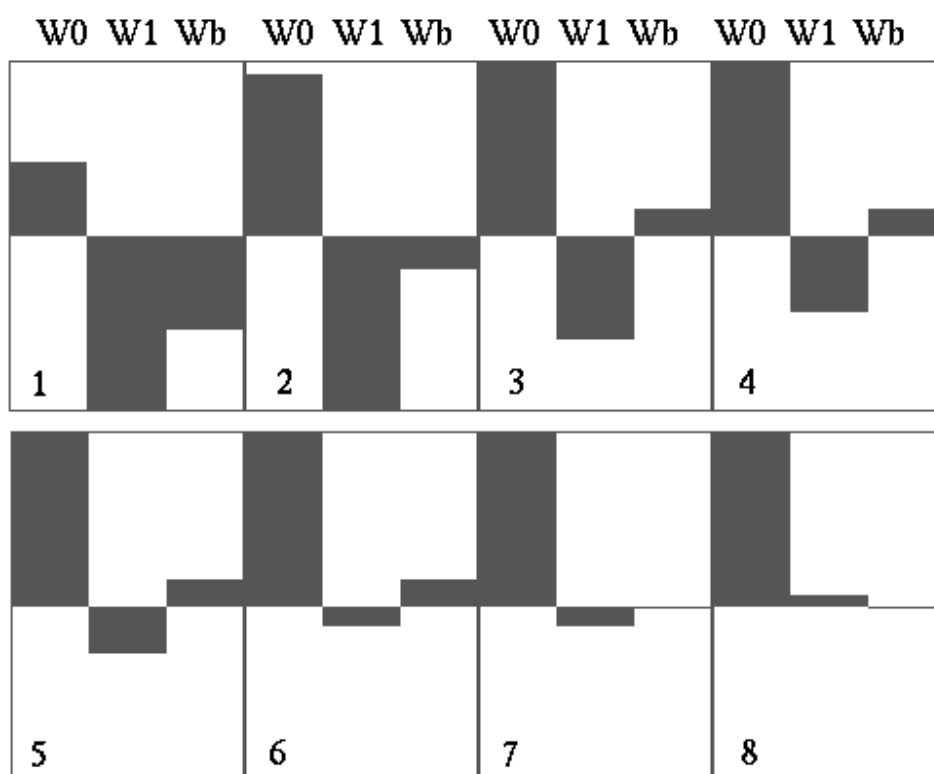
$$\Delta W_i = \eta * (D - Y) \cdot I_i$$

where η is the learning rate, D is the desired output, and Y is the actual output.

This is called the *Perceptron Learning Rule*, and goes back to the early 1960's.

We expose the net to the patterns:

I_0	I_1	Desired output
0	0	0
0	1	1
1	0	1
1	1	1



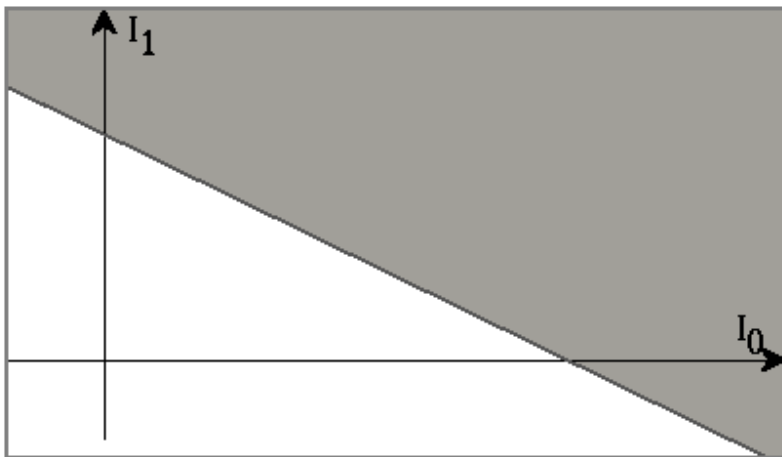
We *train* the network on these examples. Weights after each epoch (exposure to complete set of patterns)

At this point (8) the network has finished learning. Since (D-

$Y=0$ for all patterns, the weights cease adapting. Single perceptrons are limited in what they can learn:

If we have two inputs, the decision surface is a line. ... and its equation is

$$I_1 = (W_0/W_1) \cdot I_0 + (W_b/W_1)$$



In general, they implement a simple hyperplane decision surface

This restricts the possible mappings available.

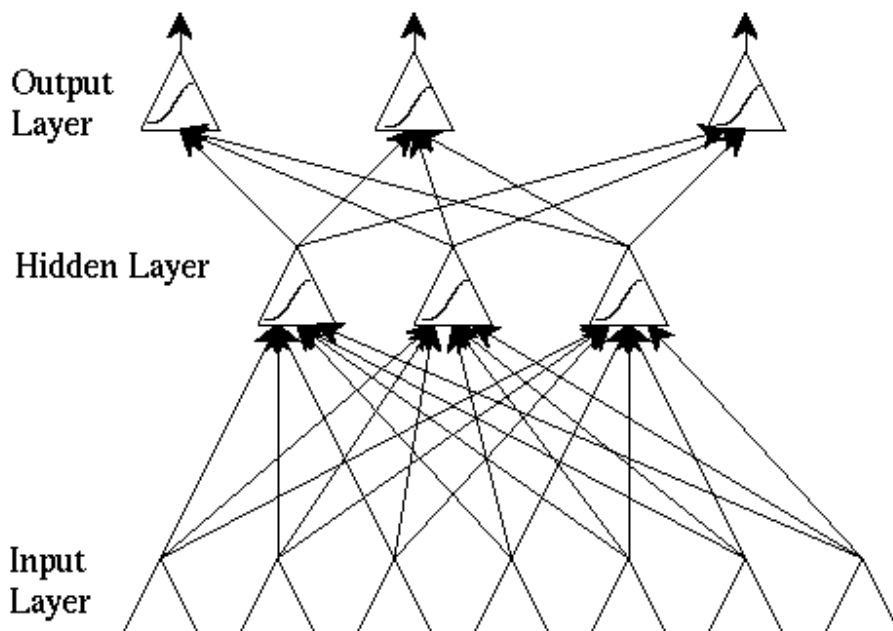
Developments from the simple perceptron:

Back-Propagated Delta Rule Networks (BP)

(sometimes known as multi-layer perceptrons (MLPs)) and Radial Basis Function Networks (RBF) are both well-known developments of the Delta rule for single layer networks (itself a development of the Perceptron Learning Rule). Both can learn arbitrary mappings or classifications. Further, the inputs (and outputs) can have real values

Back-Propagated Delta Rule Networks (BP)

is a development from the simple Delta rule in which extra *hidden layers* (layers additional to the input and output layers, not connected externally) are added. The network topology is constrained to be *feedforward*: i.e. loop-free - generally connections are allowed from the input layer to the first (and possibly only) hidden layer; from the first hidden layer to the second, ..., and from the last hidden layer to the output layer.



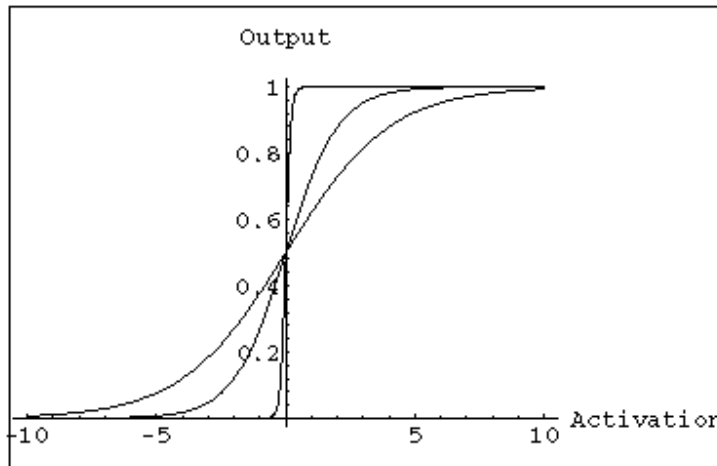
Typical BP network architecture:

The hidden layer learns to *recode* (or to *provide a representation* for) the inputs. More than one

hidden layer can be used.

The architecture is more powerful than single-layer networks: it can be shown that any mapping can be learned, given two hidden layers (of units).

The units are a little more complex than those in the original perceptron: their input/output graph is



As a function:

$$Y = 1 / (1 + \exp(-k \cdot (\sum W_{in} * X_{in})))$$

The graph shows the output for $k=0.5$, 1 , and 10 , as the activation varies from -10 to 10 .

Training BP Networks

The weight change rule is a development of the perceptron

learning rule. Weights are changed by an amount proportional to *the error at that unit times the output of the unit feeding into the weight.*

Running the network consists of

Forward pass:

the outputs are calculated and the error at the output units calculated.

Backward pass:

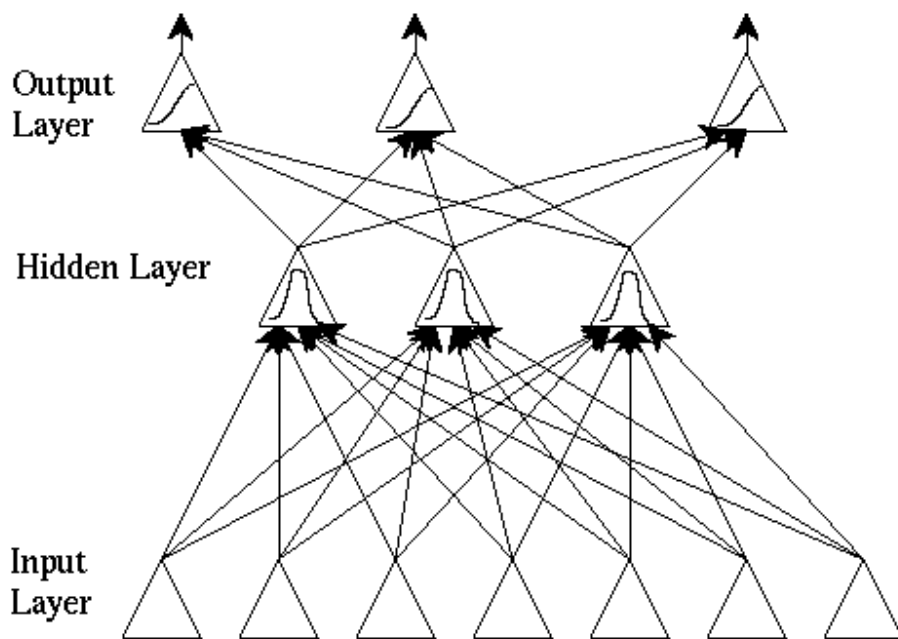
The output unit error is used to alter weights on the output units. Then the error at the hidden nodes is calculated (by *back-propagating* the error at the output units through the weights), and the weights on the hidden nodes altered using these values.

For each data pair to be learned a forward pass and backwards pass is performed. This is repeated over and over again until the error is at a low enough level (or we give up).

Radial Basis function Networks

Radial basis function networks are also feedforward, but have only *one* hidden layer.

Typical RBF architecture:

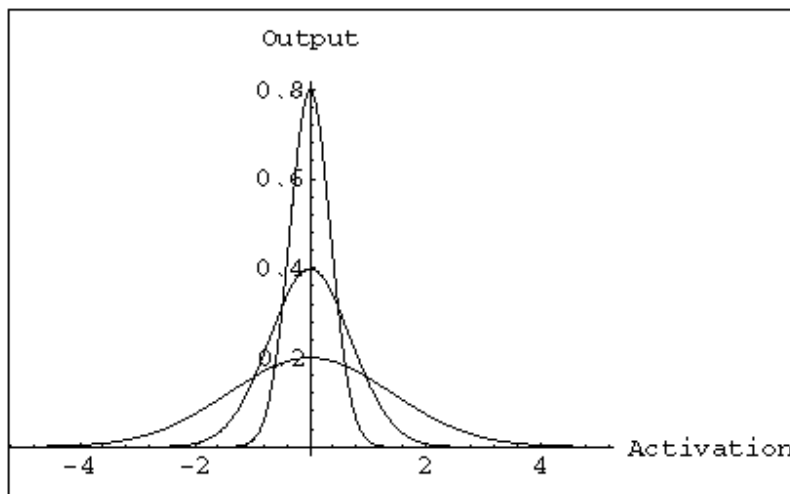


Like BP, RBF nets can learn arbitrary mappings: the primary difference is in the hidden layer.

RBF hidden layer units have a *receptive field* which has a *centre*: that is, a particular input value at which they have a

maximal output. Their output tails off as the input moves away from this point.

Generally, the hidden unit function is a Gaussian:



Gaussians with three different standard deviations.

Training RBF Networks.

RBF networks are trained by

- deciding on how many hidden units there should be
- deciding on their centres and the

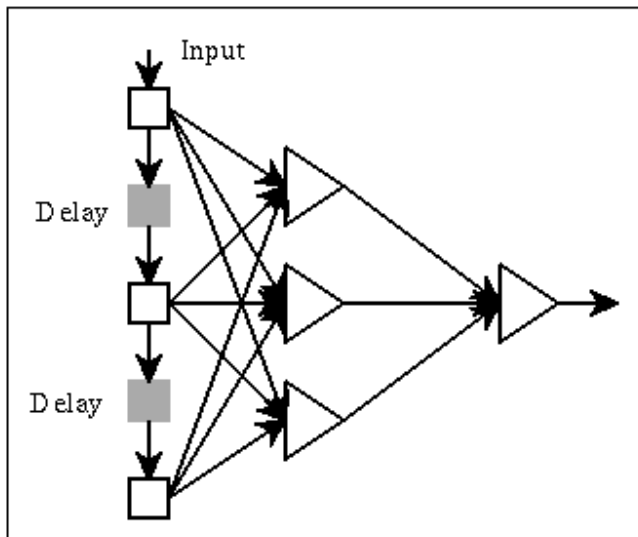
- sharpnesses (standard deviation) of their Gaussians
- training up the output layer.

Generally, the centres and SDs are decided on first by examining the vectors in the training data. The output layer weights are then trained using the Delta rule. BP is the most widely applied neural network technique. RBFs are gaining in popularity.

Nets can be

- trained on classification data (each output represents one class), and then used directly as classifiers of new data.
- trained on $(x, f(x))$ points of an unknown function f , and then used to interpolate.

RBFs have the advantage that one can add extra units with centres near parts of the input which are difficult to classify. Both BP and RBFs can also be used for processing time-varying data: one can consider a *window* on the data:



Networks of this form (finite-impulse response) have been used in many applications.

There are also networks whose architectures are specialised for processing time-series.

Unsupervised networks:

Simple Perceptrons, BP, and RBF networks need a teacher to tell the network what the desired output should be. These are supervised networks.

In an unsupervised net, the network adapts purely in response to its inputs. Such networks can learn to pick out structure in their input.

Applications for unsupervised nets

clustering data:

exactly one of a small number of output units comes on in response to an input.

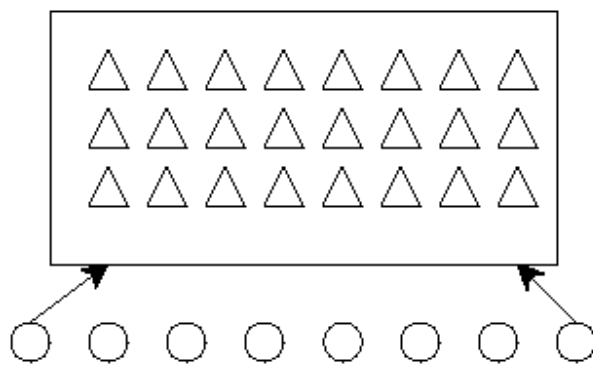
reducing the dimensionality of data:

data with high dimension (a large number of input units) is compressed into a lower dimension (small number of output units).

Although learning in these nets can be slow, running the trained net is very fast - even on a computer simulation of a neural net.

Kohonen clustering Algorithm:

- takes a high-dimensional input, and clusters it, but retaining some topological ordering of the output.



**2-dimensional
array of
output units**

After training, an input will cause *some* the output units in some area to become active.

**High
dimensional
input**

Such clustering (and dimensionality reduction) is very useful as a preprocessing

stage, whether for further neural network data processing, or for more traditional techniques.

Where are Neural Networks applicable?

..... or are they just a solution in search of a problem?

Neural networks cannot do anything that cannot be done using traditional computing techniques, BUT they can do some things which would otherwise be very difficult.

In particular, they can form a model from their training data (or possibly input data) alone.

This is particularly useful with sensory data, or with data from a complex (e.g. chemical, manufacturing, or commercial) process. There may be an algorithm, but it is not known, or has too many variables. It is easier to let the network learn from examples.

Neural networks are being used:

- in investment analysis:
to attempt to predict the movement of stocks currencies etc., from previous data. There, they are replacing earlier simpler linear models.
- in signature analysis:
as a mechanism for comparing signatures made (e.g. in a bank) with those stored. This is one of the first large-scale applications of neural networks in the USA, and is also one of the first to use a neural network chip.
- in process control:
there are clearly applications to be made here: most processes cannot be determined as computable algorithms. Newcastle University Chemical Engineering Department is working with industrial partners (such as Zeneca and BP) in this area.
- in monitoring:
networks have been used to monitor

- the state of aircraft engines. By monitoring vibration levels and sound, early warning of engine problems can be given.
- British Rail have also been testing a similar application monitoring diesel engines.
- in marketing:
networks have been used to improve marketing mailshots. One technique is to run a test mailshot, and look at the pattern of returns from this. The idea is to find a predictive mapping from the data known about the clients to how they have responded. This mapping is then used to direct further mailshots.

Where are neural networks going?

A great deal of research is going on in neural networks worldwide.

This ranges from basic research into new and more efficient learning algorithms, to networks which can respond to temporally varying patterns (both ongoing at Stirling), to techniques for implementing neural networks directly in silicon. Already one chip commercially available exists, but it does not include adaptation. Edinburgh University have implemented a neural network chip, and are working on the learning problem.

Production of a learning chip would allow the application of this technology to a whole range of problems where the price of a PC and software cannot be justified.

There is particular interest in sensory and sensing applications: nets which learn to interpret real-world sensors and learn about their environment.

New Application areas:

- Pen PC's
PC's where one can write on a tablet, and the writing will be recognised and translated into (ASCII) text.
- Speech and Vision recognition systems
Not new, but Neural Networks are becoming increasingly part of such systems. They are used as a system component, in conjunction with traditional computers.
- White goods and toys
As Neural Network chips become available, the possibility of simple cheap systems which have learned to recognise simple entities (e.g. walls looming, or simple commands like Go, or Stop), may lead to their incorporation in toys and washing machines etc. Already the Japanese are using a related technology, fuzzy logic, in this way. There is considerable interest in the combination of fuzzy and neural technologies.

Some comments (added September 2001)

Reading this through, it is a bit outdated: not that there's anything incorrect above, but the world has moved on. Neural Networks should be seen as part of a larger field sometimes

called *Soft Computing* or *Natural Computing*. In the last few years, there has been a real movement of the discipline in three different directions:

Neural networks, statistics, generative models, Bayesian inference

There is a sense in which these fields are coalescing. The real problem is making conclusions from incomplete, noisy data, and all of these fields offer something in this area. Developments in the mathematics underlying these fields have shown that there are real similarities in the techniques used. Chris Bishop's book *Neural Networks for Pattern Recognition*, Oxford University Press is a good start on this area.

Neuromorphic Systems

Existing neural network (and indeed other soft computing) systems are generally software models for solving static problems on PCs. But why not free the concept from the workstation? The area of neuromorphic systems is concerned with real-time implementations of neurally inspired systems, generally implemented directly in silicon, for sensory and motor tasks. Another aspect is direct implementation of detailed aspects of neurons in silicon (see *Biological Neural Networks* below). The main centres worldwide are at the Institute for neuroinformatics at Zurich, and at the Center for Neuromorphic Systems Engineering at Caltech.

Biological Neural Networks

There is real interest in how neural network research and neurophysiology can come together. The pattern recognition aspects of Artificial Neural Networks don't really explain too much about how real brains actually work. The field called Computational Neuroscience has taken inspiration from both artificial neural networks and neurophysiology, and attempts to put the two together.

FUZZY LOGIC

Fuzzy logic is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic. It can be thought of as the application side of fuzzy set theory dealing with well thought out real world expert values for a complex problem (Klir 1997).

Degrees of truth are often confused with probabilities. However, they are distinct conceptually; fuzzy truth represents membership in vaguely defined sets, not likelihood of some event or condition. For example, if a 100-ml glass contains 30 ml of water, then, for two fuzzy sets, Empty and Full, one might define the glass as being 0.7 empty and 0.3 full. Note that the concept of emptiness would be subjective and thus would depend on the observer or designer. Another designer might equally well design a set membership function where the glass would be considered full for all values down to 50 ml. A probabilistic setting would first define a scalar variable for the fullness of the glass, and second, conditional distributions describing the probability that someone would call the glass full given a specific fullness level. Note that the conditioning can be achieved by

having a specific observer that randomly selects the label for the glass, a distribution over deterministic observers, or both. While fuzzy logic avoids talking about randomness in this context, this simplification at the same time obscures what is exactly meant by the statement the 'glass is 0.3 full'.

Fuzzy logic allows for set membership values to range (inclusively) between 0 and 1, and in its linguistic form, imprecise concepts like "slightly", "quite" and "very". Specifically, it allows partial membership in a set. It is related to fuzzy sets and possibility theory. It was introduced in 1965 by Lotfi Zadeh at the University of California, Berkeley.

Fuzzy logic is controversial in some circles and is rejected by some control engineers and by most statisticians who hold that probability is the only rigorous mathematical description of uncertainty. Critics also argue that it cannot be a superset of ordinary set theory since membership functions are defined in terms of conventional sets.

Applications

Fuzzy logic can be used to control household appliances such as washing machines (which sense load size and detergent concentration and adjust their wash cycles accordingly) and refrigerators.

A basic application might characterize subranges of a continuous variable. For instance, a temperature measurement for anti-lock brakes might have several separate membership functions defining particular temperature ranges needed to control the brakes properly. Each function maps the same temperature value to a truth value in the 0 to 1 range. These truth values can then be used to determine how the brakes should be controlled.

In this image, *cold*, *warm*, and *hot* are functions mapping a temperature scale. A point on that scale has three "truth values" — one for each of the three functions. For the particular temperature illustrated with the vertical line, the three truth values (0.8, 0.2, and 0) could be interpreted as evaluating that particular temperature as being, say, "fairly cold" (blue arrow), "slightly warm" (yellow arrow), and "not hot" (red arrow).

Misconceptions and controversies

Fuzzy logic is the same as "imprecise logic".

Fuzzy logic is not any less precise than any other form of logic: it is an organized and mathematical method of handling *inherently* imprecise concepts. The concept of "coldness" cannot be expressed in an equation, because although temperature is a quantity, "coldness" is not. However, people have an idea of what "cold" is, and agree that there is no sharp cutoff between "cold" and "not cold", where something is "cold" at

N degrees but "not cold" at N+1 degrees — a concept classical logic cannot easily handle due to the principle of bivalence. The result has no set answer so it is believed to be a 'fuzzy' answer.

Fuzzy logic is a new way of expressing probability.

Fuzzy logic and probability are different ways of expressing uncertainty. While both fuzzy logic and probability theory can be used to represent subjective belief, fuzzy set theory uses the concept of fuzzy set membership (i.e. *how much* a variable is in a set), probability theory uses the concept of subjective probability (i.e. *how probable* do I think that a variable is in a set). While this distinction is mostly philosophical, the fuzzy-logic-derived possibility measure is inherently different from the probability measure, hence they are not *directly* equivalent. However, many statisticians are persuaded by the work of Bruno de Finetti that only one kind of mathematical uncertainty is needed and thus fuzzy logic is unnecessary. On the other hand, Bart Kosko argues that probability is a subtheory of fuzzy logic, as probability only handles one kind of uncertainty. He also claims to have proven a derivation of Bayes' theorem from the concept of fuzzy subsethood. Lotfi Zadeh, the creator of fuzzy logic, argues that fuzzy logic is different in character from probability, and is not a replacement for it. He has created a fuzzy alternative to probability, which he calls possibility theory. Other controversial approaches to uncertainty include Dempster-Shafer theory and rough sets.

Fuzzy logic will be difficult to scale to larger problems.

This criticism is mainly because there exist problems with conditional possibility, the fuzzy set theory equivalent of conditional probability. This makes it difficult to perform inference. However there have not been many studies comparing fuzzy-based systems with probabilistic ones.

Examples where fuzzy logic is used

- Automobile and other vehicle subsystems, such as automatic transmissions, ABS and cruise control (e.g. Tokyo monorail)
- Air conditioners
- The Massive engine used in the Lord of the Rings films, which helped show huge scale armies create random, yet orderly movements
- Cameras
- Digital image processing, such as edge detection

- Rice cookers
- Dishwashers
- Elevators
- Washing machines and other home appliances
- Video game artificial intelligence
- Language filters on message boards and chat rooms for filtering out offensive text
- Pattern recognition in Remote Sensing
- Fuzzy logic has also been incorporated into some microcontrollers and microprocessors, for instance, the Freescale 68HC12.

How fuzzy logic is applied

Fuzzy Set Theory defines Fuzzy Operators on Fuzzy Sets. The problem in applying this is that the appropriate Fuzzy Operator may not be known. For this reason, Fuzzy logic usually uses IF/THEN rules, or constructs that are equivalent, such as fuzzy associative matrices.

Rules are usually expressed in the form:
 IF *variable* IS *set* THEN *action*

For example, an extremely simple temperature regulator that uses a fan might look like this:

IF temperature IS very cold THEN stop fan
 IF temperature IS cold THEN turn down fan
 IF temperature IS normal THEN maintain level
 IF temperature IS hot THEN speed up fan

Notice there is no "ELSE". All of the rules are evaluated, because the temperature might be "cold" and "normal" at the same time to differing degrees.

The AND, OR, and NOT operators of boolean logic exist in fuzzy logic, usually defined as the minimum, maximum, and complement; when they are defined this way, they are called the *Zadeh operators*, because they were first defined as such in Zadeh's original papers. So for the fuzzy variables x and y:

$$\text{NOT } x = (1 - \text{truth}(x))$$

$$x \text{ AND } y = \text{minimum}(\text{truth}(x), \text{truth}(y))$$

$$x \text{ OR } y = \text{maximum}(\text{truth}(x), \text{truth}(y))$$

There are also other operators, more linguistic in nature, called *hedges* that can be applied. These are generally adverbs such as "very", or "somewhat", which modify the meaning of a set using a mathematical formula.

In application, the programming language Prolog is well geared to implementing fuzzy logic with its facilities to set up a database of "rules" which are queried to deduct logic. This sort of programming is known as logic programming.

Once fuzzy relations are defined, it is possible to develop fuzzy relational databases. The first fuzzy relational database, FRDB, appeared in Maria Zemankova's dissertation. After, some other models arose like the Buckles-Petry model, the Prade-Testemale Model, the Umamo-Fukami model or the GEFRED model by J.M. Medina, M.A. Vila et al. In the context of fuzzy databases, some fuzzy querying languages have been defined, highlighting the SQLf by P. Bosc et al. and the FSQl by J. Galindo et al. These languages define some structures in order to include fuzzy aspects in the SQL statements, like fuzzy conditions, fuzzy comparators, fuzzy constants, fuzzy constraints, fuzzy thresholds, linguistic labels and so on.

Other examples

If a man is 1.8 meters, consider him as tall:

IF male IS true AND height \geq 1.8 THEN is_tall IS true; is_short IS false

The fuzzy rules do not make the sharp distinction between tall and short, that is not so realistic:

IF height \leq medium male THEN is_short IS agree somewhat
 IF height \geq medium male THEN is_tall IS agree somewhat

In the fuzzy case, there are no such heights like 1.83 meters, but there are fuzzy values, like the following assignments:

dwarf	male	=	[0,	1.3]	m
short	male	=	(1.3,	1.5]	1.5]
medium	male	=	(1.5,	1.8]	1.8]
tall	male	=	(1.8,	2.0]	2.0]
giant male $>$ 2.0 m					

For the consequent, there are also not only two values, but five, say:

agree	not	=	0
agree	little	=	1
agree	somewhat	=	2

Rational Pavelka logic is a generalization of multi-valued logic. It is an extension of Łukasiewicz fuzzy logic with additional constants.

All these logics encompass the traditional propositional logic (whose models correspond to Boolean algebras).

Predicate fuzzy logics

These extend the above-mentioned fuzzy logics by adding universal and existential quantifiers in a manner similar to the way that predicate logic is created from propositional logic. The semantics of the universal resp. existential quantifier in t-norm fuzzy logics is the infimum resp. supremum of the truth degrees of the instances of the quantified subformula.

Effectiveness for fuzzy logics

The notions of a "decidable subset" and "recursively enumerable subset" are basic ones for classical mathematics and classical logic. Then, the question of a suitable extension of such concepts to fuzzy set theory arises. A first proposal in such a direction was made by E.S. Santos by the notions of *fuzzy Turing machine*, *Markov normal fuzzy algorithm* and *fuzzy program*. Successively, L. Biacino and G. Gerla proposed the following definition where \dot{U} denotes the set of rational numbers in $[0,1]$. A fuzzy subset $\mu : S \rightarrow [0,1]$ of a set S is *recursively enumerable* if a recursive map $h : S \times \mathbb{N} \rightarrow \dot{U}$ exists such that, for every x in S , the function $h(x,n)$ is increasing with respect to n and $\mu(x) = \lim h(x,n)$. We say that μ is *decidable* if both μ and its complement $-\mu$ are recursively enumerable. An extension of such a theory to the general case of the L-subsets is proposed in Gerla 2006. The proposed definitions are well related with fuzzy logic. Indeed, the following theorem holds true (provided that the deduction apparatus of the fuzzy logic satisfies some obvious effectiveness property).

Theorem. Any axiomatizable fuzzy theory is recursively enumerable. In particular, the fuzzy set of logically true formulas is recursively enumerable in spite of the fact that the crisp set of valid formulas is not recursively enumerable, in general. Moreover, any axiomatizable and complete theory is decidable.

It is an open question to give supports for a *Church thesis* for fuzzy logic claiming that the proposed notion of recursive enumerability for fuzzy subsets is the adequate one. To this aim, further investigations on the notions of fuzzy grammar and fuzzy Turing machine should be necessary (see for example Wiedermann's paper). Another open question is to start from this notion to find an extension of Gödel's theorems to fuzzy logic.

GENETIC ALGORITHMS

A genetic algorithm (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Genetic algorithms find application in bioinformatics, phylogenetics, computer science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.

A typical genetic algorithm requires two things to be defined:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in Genetic programming and graph-form representations are explored in Evolutionary programming.

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem we want to maximize the total value of objects that we can put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the

solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Once we have the genetic representation and the fitness function defined, GA proceeds to initialize a population of solutions randomly, then improve it through repetitive application of mutation, crossover, inversion and selection operators.

Initialization

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the *search space*). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

Reproduction

The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each child, and the process continues until a new population of solutions of appropriate size is generated.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first

generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above.

Pseudo-code algorithm

- Choose initial population
- Evaluate the fitness of each individual in the population
- Repeat
- Select best-ranking individuals to reproduce
- Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
- Evaluate the individual fitnesses of the offspring
- Replace worst ranked part of population with offspring
- Until termination

Observations

- There are several general observations about the generation of solutions via a genetic algorithm:
- In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short-term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum, others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, although the No Free Lunch theorem proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be

maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Diversity is important in genetic algorithms (and genetic programming) because crossing over a homogeneous population does not yield new solutions. In evolution strategies and evolutionary programming, diversity is not essential because of a greater reliance on mutation.

- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called *triggered hypermutation*), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called *random immigrants*). Recent research has also shown the benefits of using biological exaptation (or preadaptation) in solving this problem.¹ Again, evolution strategies and evolutionary programming can be implemented with a so-called "comma strategy" in which parents are not maintained and new parents are selected only from offspring. This can be more effective on dynamic problems.
- GAs cannot effectively solve problems in which the only fitness measure is right/wrong, as there is no way to converge on the solution. (No hill to climb.) In these cases, a random search may find a solution as quickly as a GA.
- Selection is clearly an important genetic operator, but opinion is divided over the importance of crossover versus mutation. Some argue that crossover is the most important, while mutation is only necessary to ensure that potential solutions are not lost. Others argue that crossover in a largely uniform population only serves to propagate innovations originally found by mutation, and in a non-uniform population crossover is nearly always equivalent to a very large mutation (which is likely to be catastrophic). There are many references in Fogel (2006) that support the importance of mutation-based search, but across all problems the No Free Lunch theorem holds, so these opinions are without merit unless the discussion is restricted to a particular problem.
- Often, GAs can rapidly locate *good* solutions, even for difficult search spaces. The same is of course also true for evolution strategies and evolutionary programming.
- For specific optimization problems and problem instantiations, simpler optimization algorithms may find better solutions than genetic algorithms (given the same amount of computation time). Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and swarm intelligence (e.g.: ant colony optimization, particle swarm optimization).
- As with all current machine learning problems it is worth tuning the parameters such as mutation probability, recombination probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions unless there is elitist selection. There are theoretical but not yet practical upper and lower bounds for these parameters that can help guide selection.

- The implementation and evaluation of the fitness function is an important factor in the speed and efficiency of the algorithm.

Variants

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point representations. The floating point representation is natural to evolution strategies and evolutionary programming. The notion of real-valued genetic algorithms has been offered but is really a misnomer because it does not really represent the building block theory that was proposed by Holland in the 1970s. This theory is not without support though, based on theoretical and experimental results (see below). The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

When bit strings representations of integers are used, Gray coding is often employed. In this way, small changes in the integer can be readily effected through mutations or crossovers. This has been found to help prevent premature convergence at so called *Hamming walls*, in which too many simultaneous mutations (or crossover events) must occur in order to change the chromosome to a better solution.

Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes. Theoretically, the smaller the alphabet, the better the performance, but paradoxically, good results have been obtained from using real-valued chromosomes.

A very successful (slight) variant of the general process of constructing a new population is to allow some of the better organisms from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection*.

Parallel implementations of genetic algorithms come in two flavours. Coarse grained parallel genetic algorithms assume a population on each of the computer nodes and migration of individuals among the nodes. Fine grained parallel genetic algorithms assume an individual on each processor node which acts with neighboring individuals for selection and reproduction. Other variants, like genetic algorithms for online optimization problems, introduce time-dependence or noise in the fitness function.

It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region. Alternating

GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

An algorithm that maximizes mean fitness (without any need for the definition of mean fitness as a criterion function) is Gaussian adaptation. See Kjellström 1970^[21], provided that the ontogeny of an individual may be seen as a modified recapitulation of evolutionary random steps in the past and that the sum of many random steps tend to become Gaussian distributed (according to the central limit theorem).

This means that the rules of genetic variation may have a different meaning in the natural case. For instance - provided that steps are stored in consecutive order - crossing over may sum a number of steps from maternal DNA adding a number of steps from paternal DNA and so on. This is like adding vectors that more probably may follow a ridge in the phenotypic landscape. Thus, the efficiency of the process may be increased by many orders of magnitude. Moreover, the inversion operator has the opportunity to place steps in consecutive order or any other suitable order in favour of survival or efficiency.

Gaussian adaptation is able to approximate the natural process by an adaptation of the moment matrix of the Gaussian. So, because very many quantitative characters are Gaussian distributed in a large population, Gaussian adaptation may serve as a genetic algorithm replacing the rules of genetic variation by a Gaussian random number generator working on the phenotypic level.

Population-based incremental learning is a variation where the population as a whole is evolved rather than its individual members.

Problem domains

Problems which appear to be particularly appropriate for solution by genetic algorithms include timetabling and scheduling problems, and many scheduling software packages are based on GAs. GAs have also been applied to engineering. Genetic algorithms are often applied as an approach to solve global optimization problems.

As a general rule of thumb genetic algorithms might be useful in problem domains that have a complex fitness landscape as recombination is designed to move the population away from local optima that a traditional hill climbing algorithm might get stuck in.

History

Computer simulations of evolution started as early as in 1954 with the work of Nils Aall Barricelli, who was using the computer at the Institute for Advanced Study in Princeton, New Jersey. His 1954 publication was not widely noticed. Starting in 1957, the Australian quantitative geneticist Alex Fraser published a series of papers on simulation of artificial selection of organisms with multiple loci controlling a measurable trait. From these beginnings, computer simulation of evolution by biologists became more common in the early 1960s, and the methods were described in books by Fraser and Burnell (1970)

and Crosby (1973). Fraser's simulations included all of the essential elements of modern genetic algorithms. In addition, Hans Bremermann published a series of papers in the 1960s that also adopted a population of solution to optimization problems, undergoing recombination, mutation, and selection. Bremermann's research also included the elements of modern genetic algorithms. Other noteworthy early pioneers include Richard Friedberg, George Friedman, and Michael Conrad. Many early papers are reprinted by Fogel (1998).

Although Barricelli, in work he reported in 1963, had simulated the evolution of ability to play a simple game, artificial evolution became a widely recognized optimization method as a result of the work of Ingo Rechenberg and Hans-Paul Schwefel in the 1960s and early 1970s - his group was able to solve complex engineering problems through evolution strategies. Another approach was the evolutionary programming technique of Lawrence J. Fogel, which was proposed for generating artificial intelligence. Evolutionary programming originally used finite state machines for predicting environments, and used variation and selection to optimize the predictive logics. Genetic algorithms in particular became popular through the work of John Holland in the early 1970s, and particularly his book *Adaptation in Natural and Artificial Systems* (1975). His work originated with studies of cellular automata, conducted by Holland and his students at the University of Michigan. Holland introduced a formalized framework for predicting the quality of the next generation, known as Holland's Schema Theorem. Research in GAs remained largely theoretical until the mid-1980s, when The First International Conference on Genetic Algorithms was held in Pittsburgh, Pennsylvania.

As academic interest grew, the dramatic increase in desktop computational power allowed for practical application of the new technique. In the late 1980s, General Electric started selling the world's first genetic algorithm product, a mainframe-based toolkit designed for industrial processes. In 1989, Axcelis, Inc. released Evolver, the world's second GA product and the first for desktop computers. The New York Times technology writer John Markoff wrote¹ about Evolver in 1990.

Related techniques

- Ant colony optimization (ACO) uses many ants (or agents) to traverse the solution space and find locally productive areas. While usually inferior to genetic algorithms and other forms of local search, it is able to produce results in problems where no global or up-to-date perspective can be obtained, and thus the other methods cannot be applied.
- Bacteriologic Algorithms (BA) inspired by evolutionary ecology and, more particularly, bacteriologic adaptation. Evolutionary ecology is the study of living organisms in the context of their environment, with the aim of discovering how they adapt. Its basic concept is that in a heterogeneous environment, you can't find one individual that fits the whole environment. So, you need to reason at the population level. BAs have shown better results than GAs on problems such as complex positioning problems (antennas for cell phones, urban planning, and so on) or data mining.

- Cross-entropy method The Cross-entropy (CE) method generates candidate solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.
- Cultural algorithm (CA) consists of the population component almost identical to that of the genetic algorithm and, in addition, a knowledge component called the belief space.
- Evolution strategies (ES, see Rechenberg, 1971) evolve individuals by means of mutation and intermediate and discrete recombination. ES algorithms are designed particularly to solve problems in the real-value domain. They use self-adaptation to adjust control parameters of the search.
- Evolutionary programming (EP) involves populations of solutions with primarily mutation and selection and arbitrary representations. They use self-adaptation to adjust parameters, and can include other variation operations such as combining information from multiple parents.
- Extremal optimization (EO) Unlike GAs, which work with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to be assigned a quality measure ("fitness"). The governing principle behind this algorithm is that of *emergent* improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is decidedly at odds with a GA that selects good solutions in an attempt to make better solutions.
- Gaussian adaptation (normal or natural adaptation, abbreviated NA to avoid confusion with GA) is intended for the maximisation of manufacturing yield of signal processing systems. It may also be used for ordinary parametric optimisation. It relies on a certain theorem valid for all regions of acceptability and all Gaussian distributions. The efficiency of NA relies on information theory and a certain theorem of efficiency. Its efficiency is defined as information divided by the work needed to get the information. Because NA maximises mean fitness rather than the fitness of the individual, the landscape is smoothed such that valleys between peaks may disappear. Therefore it has a certain "ambition" to avoid local peaks in the fitness landscape. NA is also good at climbing sharp crests by adaptation of the moment matrix, because NA may maximise the disorder (average information) of the Gaussian simultaneously keeping the mean fitness constant.
- Genetic programming (GP) is a related technique popularized by John Koza in which computer programs, rather than function parameters, are optimized. Genetic programming often uses tree-based internal data structures to represent the computer programs for adaptation instead of the list structures typical of genetic algorithms.
- Grouping Genetic Algorithm (GGA) is an evolution of the GA where the focus is shifted from individual items, like in classical GAs, to groups or subset of items. The idea behind this GA evolution proposed by Emanuel Falkenauer is that solving some complex problems, a.k.a. *clustering* or *partitioning* problems where a set of items must be split into disjoint group of items in an optimal way, would

better be achieved by making characteristics of the groups of items equivalent to genes. These kind of problems include Bin Packing, Line Balancing, Clustering w.r.t. a distance measure, Equal Piles, etc., on which classic GAs proved to perform poorly. Making genes equivalent to groups implies chromosomes that are in general of variable length, and special genetic operators that manipulate whole groups of items. For Bin Packing in particular, a GGA hybridized with the Dominance Criterion of Martello and Toth, is arguably the best technique to date.

- Harmony search (HS) is an algorithm mimicking musicians behaviors in improvisation process.
- Interactive evolutionary algorithms are evolutionary algorithms that use human evaluation. They are usually applied to domains where it is hard to design a computational fitness function, for example, evolving images, music, artistic designs and forms to fit users' aesthetic preference.
- Memetic algorithm (MA), also called *hybrid genetic algorithm* among others, is a relatively new evolutionary method where local search is applied during the evolutionary cycle. The idea of memetic algorithms comes from memes, which—unlike genes—can adapt themselves. In some problem areas they are shown to be more efficient than traditional evolutionary algorithms.
- Simulated annealing (SA) is a related global optimization technique that traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation that lowers fitness is accepted probabilistically based on the difference in fitness and a decreasing temperature parameter. In SA parlance, one speaks of seeking the lowest energy instead of the maximum fitness. SA can also be used within a standard GA algorithm by starting with a relatively high rate of mutation and decreasing it over time along a given schedule.
- Stochastic optimization is an umbrella set of methods that includes GAs and numerous other approaches.
- Tabu search (TS) is similar to Simulated Annealing in that both traverse the solution space by testing mutations of an individual solution. While simulated annealing generates only one mutated solution, tabu search generates many mutated solutions and moves to the solution with the lowest energy of those generated. In order to prevent cycling and encourage greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains elements of the tabu list, which is updated as the solution traverses the solution space.

Building block hypothesis

- Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular it is difficult to understand why they are often successful in generating solutions of high fitness. The building block hypothesis (BBH) consists of
- A description of an abstract adaptive mechanism that performs adaptation by recombining "building blocks", i.e. low order, low defining-length schemata with above average fitness.

- A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this abstract adaptive mechanism.

(Goldberg 1989:41) describes the abstract adaptive mechanism as follows:

Short, low order, and highly fit schemata are sampled, recombined [crossed over], and resampled to form strings of potentially higher fitness. In a way, by working with these particular schemata [the building blocks], we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings.

Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood [building blocks], so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks.

(Goldberg 1989) claims that the building block hypothesis is supported by Holland's schema theorem.

The building block hypothesis has been sharply criticized on the grounds that it lacks theoretical justification and experimental results have been published that draw its veracity into question. On the theoretical side, for example, Wright et al. state that

"The various claims about GAs that are traditionally made under the name of the *building block hypothesis* have, to date, no basis in theory and, in some cases, are simply incoherent"

On the experimental side uniform crossover was seen to outperform one-point and two-point crossover on many of the fitness functions studied by Syswerda. Summarizing these results, Fogel remarks that

"Generally, uniform crossover yielded better performance than two-point crossover, which in turn yielded better performance than one-point crossover"

Syswerda's results contradict the building block hypothesis because uniform crossover is extremely disruptive of short schemata whereas one and two-point crossover are more likely to conserve short schemata and combine their defining bits in children produced during recombination.

The debate over the building block hypothesis demonstrates that the issue of how GAs "work", (i.e. perform adaptation) is currently far from settled. (See the External Links section for more about this)

Applications

- Artificial Creativity
- Automated design, including research on composite material design and multi-objective design of automotive components for crashworthiness, weight savings, and other characteristics.
- Automated design of mechatronic systems using bond graphs and genetic programming (NSF).
- Automated design of industrial equipment using catalogs of exemplar lever patterns.
- Automated design of sophisticated trading systems in the financial sector.
- Building phylogenetic trees.^[21]
- Calculation of Bound states and Local-density approximations.
- Chemical kinetics (gas and solid phases)
- Configuration applications, particularly physics applications of optimal molecule configurations for particular systems like C60 (buckyballs).
- Container loading optimization.
- Code-breaking, using the GA to search large solution spaces of ciphers for the one correct decryption.^[citation needed]
- Design of water distribution systems.
- Distributed computer network topologies.
- Electronic circuit design, known as Evolvable hardware.
- File allocation for a distributed system.
- Parallelization of GAs/GPs including use of hierarchical decomposition of problem domains and design spaces nesting of irregular shapes using feature matching and GAs.
- Game Theory Equilibrium Resolution.
- Gene expression profiling analysis.^[22]
- Learning Robot behavior using Genetic Algorithms.
- Learning fuzzy rule base using genetic algorithms.
- Linguistic analysis, including Grammar Induction and other aspects of Natural Language Processing (NLP) such as word sense disambiguation.
- Marketing Mix Analysis
- Mobile communications infrastructure optimization.
- Molecular Structure Optimization (Chemistry).
- Multiple criteria production scheduling.^[23]
- Multiple population topologies and interchange methodologies.
- Multiple sequence alignment.^[24]
- Operon prediction.^[25]
- Optimisation of data compression systems, for example using wavelets.
- Protein folding and protein/ligand docking.^[26]
- Plant floor layout.
- Representing rational agents in economic models such as the cobweb model.
- RNA structure prediction.^[27]

- Scheduling applications, including job-shop scheduling. The objective being to schedule jobs in a sequence dependent or non-sequence dependent setup environment in order to maximize the volume of production while minimizing penalties such as tardiness.
- Selection of optimal mathematical model to describe biological systems.
- Software engineering
- Solving the machine-component grouping problem required for cellular manufacturing systems.
- Tactical asset allocation and international equity strategies.
- Timetabling problems, such as designing a non-conflicting class timetable for a large university.
- Training artificial neural networks when pre-classified training examples are not readily obtainable (neuroevolution).
- Traveling Salesman Problem.
- Finding hardware bugs.

The Future of AI

A Report to ARPA on Twenty-First Century Intelligent Systems identified four types of systems that will have a substantial impact on applications: intelligent simulation, intelligent information resources, intelligent project coaches, and robot teams (Grosz and Davis, 1994).

Intelligent simulations generate realistic simulated worlds that enable extensive affordable training and education that can be made available any time and anywhere. Examples may be hurricane crisis management, exploration of the impacts of different economic theories, tests of products on simulated customers, and technological design—testing features through simulation that would cost millions of dollars to test using an actual prototype.

Intelligent information resources systems (IRSS) will enable easy access to information related to a specific problem. For instance, a rural doctor whose patient presents with a rare condition might use IRSS to help assess different treatments or identify new ones. An educator might find relevant background materials, including information about similar courses taught elsewhere.

Intelligent project coaches (IPC) could function as co-workers, assisting and collaborating with design or operations teams for complex systems. Such systems could remember and recall the rationale of previous decisions and, in times of crisis, explain the methods and reasoning previously used to handle that situation. An IPC for aircraft design, for example, could enhance collaboration by keeping communication flowing among the large, distributed design staff, the program managers, the customer, and the subcontractors.

Robot teams could contribute to manufacturing by operating in a dynamic environment with minimal instrumentation, thus providing the benefits of economies of scale. They could also participate in automating sophisticated laboratory procedures that require sensing, manipulation, planning, and transport.

APPLICATION OF AI RELATED TO ENGINEERING **POWER SYSTEM CONTINGENCY SCREENING AND** **RANKING USING ARTIFICIAL NEURAL NETWORKS**

INTRODUCTION:

Contingency selection is a method to quickly identify those contingencies that may cause limit violations to reduce the number of contingencies that need to be analyzed by full AC load flow while assessing power system security. Two popular methods for contingency selection are ranking and screening methods. Ranking methods involve the ranking of contingencies in approximate order of severity. Contingencies are ranked based on the value of scalar performance index (PI)

In the ranking method the performance indices are explicitly expressed in term of network variable and directly evaluated. Screening methods use approximate network solutions such as (a) distribution factor, (b) DC load flow, (c) Linearized load flow, (d) IP_IQ method, (e) local solution method etc. To identify cases causing limit violations. The network-monitored quantities are calculated for all the contingencies and ranking is based on results of the approximate solutions. The analytical techniques are usually time consuming and therefore not always suitable for real time applications more over many PI based analytical methods suffer from the problem of misclassification or false alarm, mis classification arises when an active contingency classified as inactive. Expert systems and fuzzy system based methods are slow in execution, and they lack versatility as many expert system and fuzzy system based rules are system specific.

DRAWBACKS AND LIMITATIONS OF CONVENTIONAL METHODS:

1. These methods involve heavy computation.
2. Analytical methods are time consuming.
3. Not suitable for real time applications.
4. Problem of misclassification or / and false alarm.

Why Neural Network approach for contingency screening and ranking?

Artificial neural network methods are efficient computing models with the ability to solve nonlinear pattern matching problems. They can capture the inherent non-linearity in the input pattern and use them for classification.

Recognized shortcomings of the ANN based approach:

A large number of training samples are required for the training of the neural networks. ANNs are found to be good at interpolating within the training set, but do not extrapolate accurately outside it because of the non-linear relationship between the input and the output, and multiple correlated inputs. The problem solving knowledge of the artificial network is represented in the connection weights, and hence is difficult for a human user to calculate.

VOLTAGE SCREENING AND RANKING:

A neural network is developed for fast voltage contingency screening and ranking. The voltage performance index (PI_v) is calculated by considering the limit-violated buses only, and it is the output of the neural network. Pre contingency terminal voltages of the contingent element are selected as input features for training of the neural networks.

MW SCREENING AND RANKING:

A neural network is developed for fast MW screening and ranking. The MW performance index (PI_{mw}) is calculated by considering the limit violated lines only, and it is the output of the neural network. Pre contingency line flows of the contingent element are selected as input features for training of the neural networks.

A 3-layered feed forward neural network is used for screening and for a classification task so those PIs may be ranked in different levels.

PERFORMANCE INDICES:

System performance index is defined as a penalty function to penalize severely any violation of bus voltage constraints or line flow constraints.

The voltage performance index chosen to quantify system deficiency due to out-of limit bus voltages is defined by

$$PI_v = \sum_{i=1}^{NB} (W_{vi} / 2n) * (|V_i| - |V_i^{SP}|) / \Delta V_i^{lim})^{2n}$$

where,

$|v_i|$ -voltage magnitude at bus i

$|v_i|^{SP}$ -specified(rated)voltage magnitude at bus i

$|\Delta V_i|^{lim}$ - voltage deviation limit, above which voltage deviations are unacceptable

n- exponent of penalty function (=2 preferred)

N_B -number of buses in the system

W_i - real non-negative weighting factor (=1)

The voltage deviation Δv_i^{lim} represents the threshold above which voltage level deviations are outside their limits; any contingency load flow with voltage levels outside this limit yields a high value of the index PI_v . When all the voltage level deviations from rated the voltage is within $|\Delta v_i|^{lim}$ the voltage performance index PI_v is small. This index provides a direct means of comparing the relative severity of the different outages on the system voltage profile.

MW PERFORMANCE INDICES

An index for quantifying the extent of line overloads may be defined in terms of MW performance index:

$$PI_{MW} = \sum_{l=1}^{NL} (W_L / 2n) * (P_L / P_L^{lim})^{2n}$$

where,

P_L -the mw flow of line

P_L^{lim} -the mw capacity of line

NL -the number of lines of the system

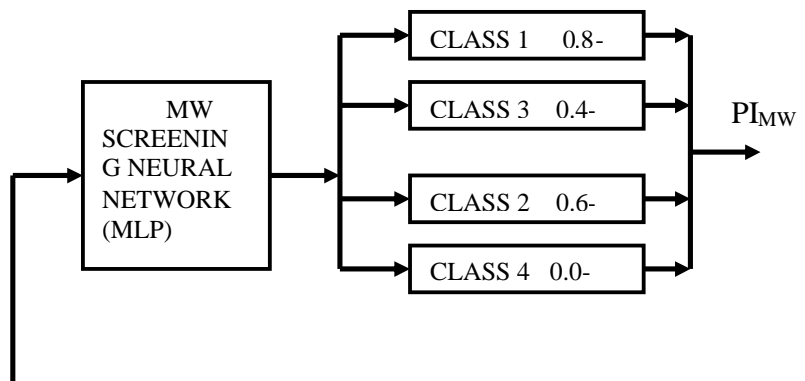
W_i -real non-negative weighting factor (=1)

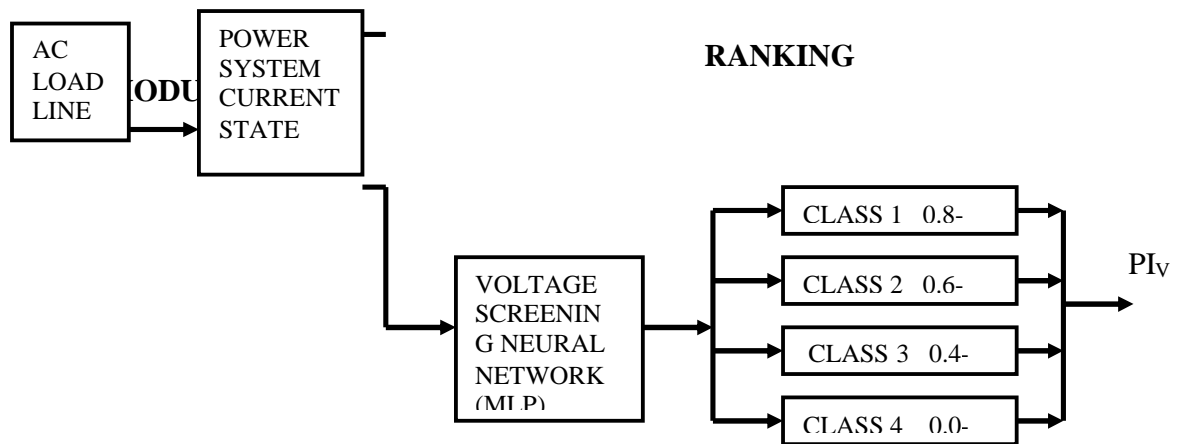
n -exponent of penalty function (=2)

This index PIMW has a small value, when all line flows are within their limits, and a high value when there are line load overloads. Thus it provides a measure of the severity of line overloads for a given state of the power system.

PROPOSED METHODOLOGY:

The block diagram of the proposed model is shown in fig.





BLOCK DIAGRAM OF VOLTAGE AND MW SCREENING AND RANKING

To obtain pre contingency line flows, terminal voltage of the contingency line as input feature training / testing of the neural network. A large number of load patterns are generated by perturbing the load on all the buses randomly in wide range of system operating conditions and a full AC load is performed for each case. Single line outage corresponding to each load pattern are also simulated by full AC load flow to calculate the PIV and PIMW

Screening module, which is three-layered feed forward ANN is used to classify the contingencies in critical or non-critical class. The screening module is trained using BP algorithm such that its target output is high (0.9) when presented with a sample from non-critical class. Ranking module consists of four modules each module is a 3-layered feed forward ANN. Class 1 has most severe contingency case (PI value above 0.8). Class 2 has PI value between 0.6 to 0.8. Class 3 has PI value between 0.4 to 0.6. Class 4 has least severe critical case (PI value below 0.4) PI value between 0.6 to 0.8.

It was observed from simulation results that for $W1=1$ and $n=2$, masking effect has been removed for the tested sample system. Here summation is carried out only for limit violated buses and lines. A number of lines or generator outages having zero value of performance index for all the load Patterns. Such lines and generators are not selected for the training of neural network. It is observed that the list of critical contingent cases is dynamic in nature depending upon the loads at various buses that is list of critical contingency may be a non critical at some other loading conditions. Similarly the ranking order of critical contingency cases may be different at different loading conditions. The input data is normalized between 0.0 and 0.9. Similarly output data

(performances indices) is normalized between (for each load pattern) 0.0 and 0.9. (0.0 for non critical case and 0.9 for most critical case for each load pattern) for some line outages, load flow solution does not converge at some loading conditions. Such line outages are placed at the ranking list with PI value at 0.9.

SUMMARY OF SOLUTION ALGORITHM:

The solution algorithm for contingency screening and ranking can be summarized in following steps:

1. A large number of load patterns are generated by perturbing loads randomly at all the buses.
2. Full AC load flow is performed for each case to compute pre outage line flows, terminal voltages at the contingent line and total load and also for single line outage to compute performance indices.
3. The line outages and the generator outages cases having zero value of PI for all the load patterns are not considered for training of neural network.
4. The input data as well as the output normalized between 0.0 and 0.9
5. The normalized input data along with PI value corresponding to the contingent line or generator are used for training of the screening module.
6. The training of the screening module is continued till the error for testing pattern is minimum only those contingency cases belonging to critical class applied to train the ranking module

RESULTS AND DISCUSSION:

The proposed neural network based method has been tested for voltage and MW contingency screening and ranking of a 6-bus system. The load patterns were generated by changing the load at each bus randomly. The full AC load flow was performed for each load pattern and for single line outages. Voltage and MW PI s were divided into 4 classes according to their severity as shown in fig. If a pattern belong to 4, it mean that the corresponding contingency is non-critical, but if it belongs to class 1 the corresponding contingency is most severe. The two graphs show's the comparison between the NR-method and proposed neural network method for 16 tested load patterns.

OBSERVATIONS:

The computation of PI MW and PI V by analytical method takes along a long time, as a load flow analysis has to be run for every single

contingency to calculate the post contingency state of the system. On the other hand, by the proposed method, once the training of the network is successfully accomplished, the test of a new contingency case is almost instantaneous. A comparison study of the result using the proposed approach and a full AC load flow analysis (using Newton raphson method) are presented in the tables.

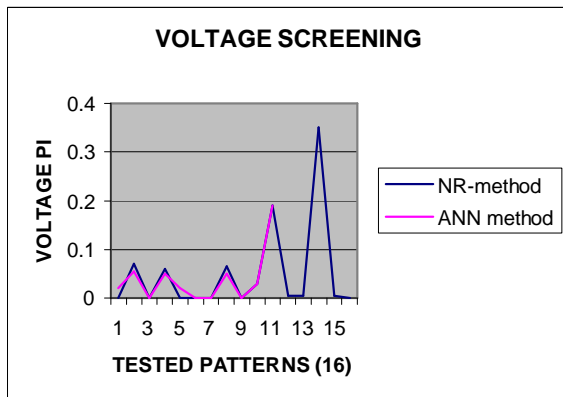
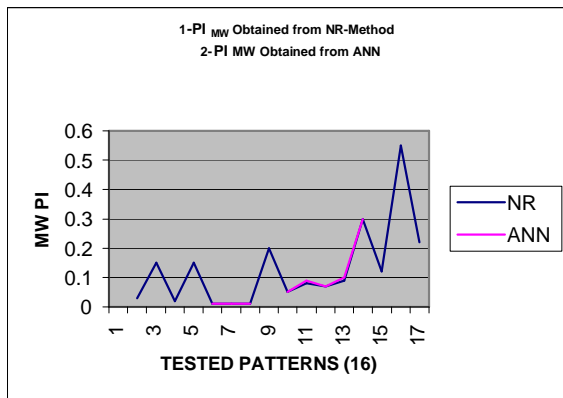
VOLTAGE SCREENING AND RANKING RESULTS

S.NO	LOAD		CONTINGENCY CASE	NR-METHOD	NR-METHOD RANKING	SCREENING MODULE OUTPUT	RANKING MODULE OUTPUT
	P	Q					
1	40	50	3-6	0.070	3	0.0595	3
2	45	45	3-6	0.059	4	0.0511	4
3	60	60	G3	0.036	5	0.0320	5
4	60	70	2-4	0.188	2	0.1859	2
5	70	70	1-4	0.005	6	0.0020	6
6	75	75	2-4	0.344	1	0.3484	1

MW SCREENING AND RANKING RESULTS

S.NO	LOAD	CONTINGENCY CASE	NR - METHOD	NR-METHOD RANKING	SCREENING MODULE OUTPUT	RANKING MODULE OUTPUT
------	------	------------------	-------------	-------------------	-------------------------	-----------------------

	P	Q					
1	40	50	3-6	0.143	3	0.1426	3
2	50	50	G3	0.024	6	0.0261	6
3	70	70	1-4	0.309	2	0.2963	2
4	75	75	2-4	0.116	5	0.1172	5
5	80	80	1-5	0.561	1	0.5462	1
6	80	80	2-6	0.137	4	0.1355	4



CONCLUSION:

Contingency screening and ranking using neural network was carried out using separate modules (neural network) for both screening and ranking. For all possible Contingencies (single Contingency) 252 patterns were generated corresponding to line and generator outages out of which only 175 (non zero value of PI) patterns were selected for training the MW Screening and ranking modules and 130 (non zero value of PI) patterns were selected for training the voltage screening and ranking modules.

AI is a young field and faces many complexities. Nonetheless, the Spring 1998 issue of *AI Magazine* contained articles on the following innovative applications of AI: This is suggestive of the broad potential of AI in the future.

REFERENCES

1. WWW.GOOGLE.COM
2. WWW.ALTAVISTA.COM

Journals.

Neural Computing and Applications, Springer-Verlag. (address: Sweetapple Ho, Catteshall Rd., Godalming, GU7 3DJ)

Books.

Fausett L., *Fundamentals of Neural Networks*, Prentice-Hall, 1994. ISBN 0 13 042250 9
or

Gurney K., *An Introduction to Neural Networks*, UCL Press, 1997, ISBN 1 85728 503 4

Haykin S., *Neural Networks*, 2nd Edition, Prentice Hall, 1999, ISBN 0 13 273350 1 is a more detailed book, with excellent coverage of the whole subject.

Submitted by

CAROL.C
MINISH.V (90982896)

ST.XAVIERS CATHOLIC COLLEGE OF ENGG.
CHUNKANKADAI