PDHonline Course E157 (3 PDH)

Artificial Intelligence: Technologies for Smart Systems Design

Instructor: Warren T. Jones, Ph.D., PE

2020

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone: 703-988-0088
www.PDHonline.com

An Approved Continuing Education Provider

Artificial Intelligence: Technologies for Smart Systems Design

Warren T. Jones, Ph.D., P.E.

Course Content

Module #1: Introduction and Definitions

What is Artificial Intelligence? The definition given by the American Association for Artificial Intelligence is "the scientific understanding of the mechanisms underlying thought and intelligent behavior and their embodiment in machines."

The term "artificial intelligence" is more likely to conjure up a vision of the science fiction robotic takeover of the earth than the more mundane and useful "smart" software functions that are becoming increasingly commonplace in engineering systems and equipment. Not surprisingly, the AI field contains elements of both of these components with the former being labeled "strong AI" and the latter "weak AI". The goal of the everyday "smart system engineer" is the building of systems that may exhibit intelligent behavior that can be leveraged to increase the productivity of the human users of the system, with little interest in the more exotic "strong AI" issues. We will not dwell here on the philosophical issues.

The genesis of the AI field is generally recognized as being the seminal paper in 1950 by Alan Turing entitled "Computing Machinery and Intelligence". Although the actual term "artificial intelligence" was not coined until six years later at a Dartmouth summer workshop, he was the first to articulate a complete vision of the field. Special purpose programming languages LISP and PROLOG were developed early in the field for non-numeric symbol processing applications development.

Module #2: Expert Systems

Rule-based Expert Systems

Expert systems technology emerged in the late 1970's when it became clear that the early AI approaches of general search mechanisms over large state spaces for solving broad classes of problems did not scale up from small problems to more complex practical applications. These disappointing results led to the realization that more practical results could be achieved building systems with narrow task-specific domain knowledge.

The name "expert systems" is derived from the term "knowledge-based expert systems". They are sometimes referred to as symbolic processors since they process linguistic symbols and lists rather that numerical data. These systems are built by capturing task-specific expertise and employ a reasoning system which imitates the human expert problem solving capabilities. Task-specific expertise is knowledge acquired from training, reading and experience and includes the following:

- Facts about the problem area

- Theoretical knowledge about the problem area

- Rigid rules and procedures regarding the problem

- "Rules of thumb" (heuristics) guidelines of what to do in specific problem situations

- Overarching strategies for problem solutions

- Problem area meta-knowledge (knowledge about knowledge)

Experts are known to make better and faster decisions in solving complex problems than non experts and becoming an expert in a given area typically takes several years. Expertise is a somewhat nebulous term and we typically refer to a person's level of expertise. The distribution of expertise appears to be the same for any type of knowledge. The upper ten percent has a performance level of about three times higher than the average and thirty times higher than the lower ten percent. This information implies that the overall effectiveness of human expertise in a given problem area can be greatly increased if the expertise of this upper ten percent can be made accessible to the others. Expert behavior seems to be characterized by the following activities:

- Recognition and problem formulation

- Rapid proper problem solving

- Explaining the solution

- Learning from experience

- Reorganizing knowledge periodically

- Knowing when to break the rules with exceptions

- Determining relevance of their expertise

- Degrading gracefully at the boundaries of their expertise

The goal of expert system development is to capture as much of the above behavior as possible. Items two and three above were the focus of early systems development. Expert systems technology represents opportunities for productivity increases in many ways. The technology can be used in the following ways:

- to capture scarce expertise

- to train future experts

- to provide 24/7 access to expertise

- to provide access in remote locations if needed

- to provide higher quality advice in some cases where the integration of the knowledge of several experts is built into the system

- to solve problems in areas that are hazardous to humans

Knowledge acquisition from experts turns out to be a complex process and is sometimes called the "bottleneck" of expert systems development. The acquired knowledge is frequently represented in the form of rules of the form IF (condition) THEN (conclusion). Hence the name Rule-Based Expert Systems.

For example,

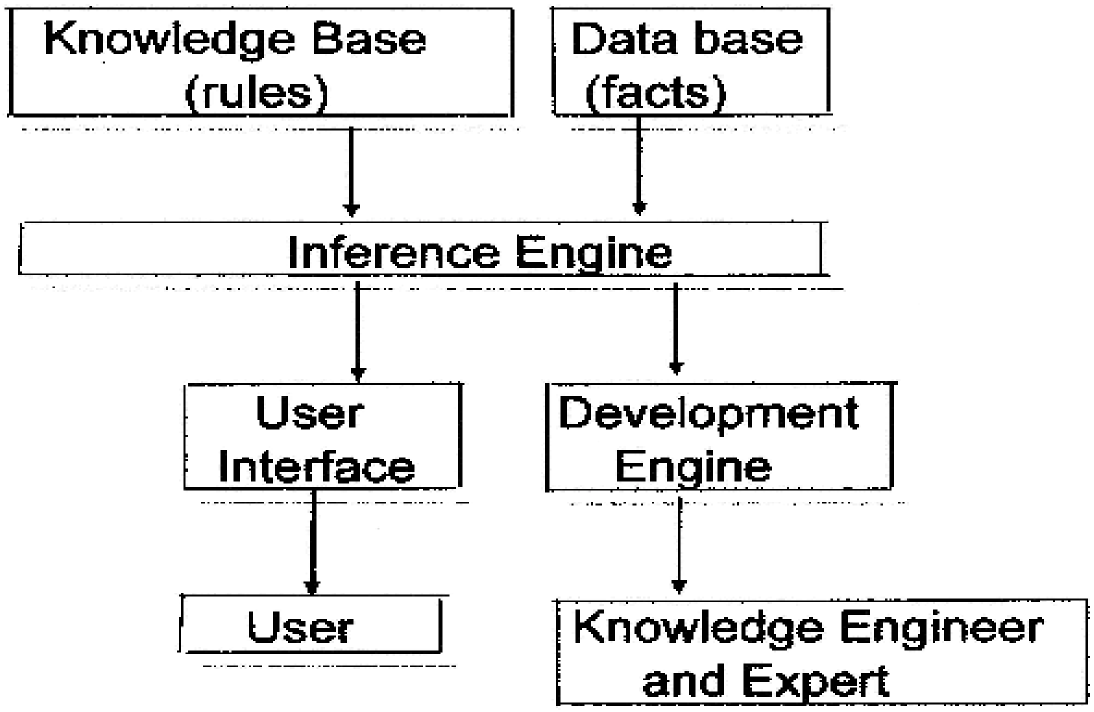
IF (car won't start), THEN (check the battery).

A knowledge base can contain hundreds or sometimes thousands of these rules. This knowledge can be obtained in a variety of ways by a knowledge engineer. Manual methods are usually some type of interviewing process designed to extract knowledge from one or more experts. There are knowledge acquisition methods approaching full automation in which rule induction systems generate rules from large collections of domain data, when available.

When completed, a knowledge base and a current collection of facts are used by the inference engine to produce a recommended solution to the problem situation represented by the input factual data. Each fact is matched to the IF part of a rule. When a match is made, the rule is said to "fire" and the THEN part of the rule is executed. This matching and firing process for the input data produces "inference chains".

Inference chains can be produced by either "forward chaining" or "backward chaining". Forward chaining is a strategy for producing whatever can be inferred from the knowledge base and the input data and is useful for analysis and interpretation. Backward chaining begins with a hypothetical conclusion and then looks for facts to support it. This strategy is usually used for diagnostic problems. Many expert system development environments support both forward and backward chaining.

The basic components of an expert system are shown below. Expert development shells, which provide all the components except for the knowledge base and database for a given application, are available from commercial vendors. With this design the same shell can be used to develop systems for many different problem domains.



Expert Systems Components

The following example will be used to illustrate the forward and backward chaining process:

Suppose the data base contains the two facts

A is true

B is true

and the following knowledge base of three rules.

Rule 1: IF A and B, THEN C

Rule 2: IF B and C, THEN D

Rule 3: IF D, THEN G

Forward chaining is data-driven and begins with the available information. Since it is known that A and B are true, it begins with one of them and searches for a rule that includes A in the IF side. It finds Rule 1 and notes that in order for this rule to fire it also needs information about B. Since B is also known to be true, Rule 1 fires and C is now added to the data base as true. Given C is now true it looks for a rule with C in the IF part. It finds Rule 2 and also notes that since B is in the data base as true, Rule 2 fires adding D to the data base as true. Given that D is true, it fires Rule 3 giving G is true.

Backward chaining is goal-driven. That is, the system begins with a hypothesis or goal in the THEN part of a rule and seeks evidence to support it. In our knowledge base the goal is G in Rule 3. The system will first check the data base to see if G is there. Since this is not the case, the system will check the IF part of Rule 3 and note that G will be true if D is true. Then it will seek a rule with a conclusion of D. This is the case for Rule 2. For D to be true in Rule 2, both B and C must be true. B is true in the data base but C is not. However, there is a Rule 1 with a conclusion of C. Since both A and B are in the database, when Rule 1 fires our evidence is now complete, and we can conclude that G is true. Many expert system development tools can provide explanatory information like the above to provide the reasoning path to the conclusion.

In this example the rules are either true or false implying the knowledge is exact. In real world situations human knowledge is frequently inexact for a variety of reasons and it is sometimes necessary to make decisions based on partial or incomplete information. Therefore, there is a need for inexact inference methods that allow the combination of imprecise data from many sources. The use of certainty factors is a popular approach to dealing with uncertainty in the knowledge base. It was first used in one of the early successful expert systems called MYCIN for the diagnosis of blood infections and meningitis.

Certainty factors provide a way for representing “degrees of belief”. The range of certainty factors is from -1 (definitely false) to +1 (definitely true). Positive values indicate the degrees of belief and negative values represent the degree of disbelief. For example, an expert might indicate that some evidence is probably true and assign a certainty factor of 0.6 to this evidence.

Certainty factors are based on the two functions:

MB(H,E), measure of belief

MD(H,E), measure of disbelief

These functions reflect the degree in which belief in hypothesis H would be increased given support for E is observed and the degree to which disbelief in hypothesis H would be reduced given the same evidence E. The range of MB(H,E) and MD(H,E) is between 0 and 1. Some data may increase the strength of belief, but conversely some data may raise the level of disbelief. The certainty factor is defined in terms of these two functions as follows:

$$cf = \frac{MB(H, E) - MD(H, E)}{1 - \min[MB(H, E), MD(H, E)]}$$

Certainty factors can be used to express the fact that a given condition may lead to more than one possible conclusion. For example,

IF A

THEN B [cf 0.8]

C [cf 0.3]

The inference engine incorporates methods for propagating certainty factors through the reasoning chain. Notice that certainty factors do not have to add up to 1.0 as in the case of probabilities.

Another way to incorporate inexact information in the system is by use of Bayesian reasoning which makes requires conditional probabilities. The Bayesian approach also requires conditional independence of evidence for both a hypothesis and its negation as well as reliable statistical data to establish prior probabilities for each hypothesis. A third approach using fuzzy expert systems is discussed in the later Fuzzy Logic section.

CLIPS (http://www.ghg.net/clips/CLIPS.html) is a popular public domain expert system building tool maintained by NASA. Jess is a rule engine for the Java^{TM} platform. (http://herzberg.ca.sandia.gov/jess/)

Expert systems have been used in a wide variety of applications including diagnosis, interpretation, prediction, design, planning, monitoring and control.

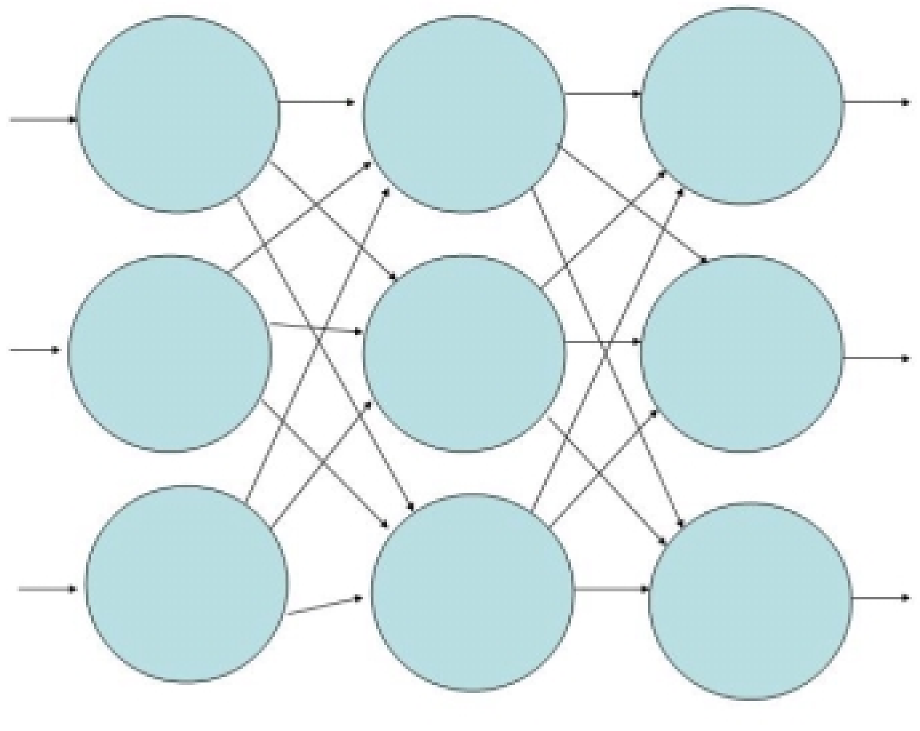
Reference

Durkin, J., **Expert Systems Design and Development,** Prentice Hall, Englewood Cliffs, NJ, 1994.

Module #3: Neural Networks

Neural networks represent a radical departure from traditional computing. They consist of networks of very small processors that are simple models of neurons in the brain. The information and knowledge of interest is stored in the dense interconnection structure, not in the discrete memory locations of traditional computing. Neural networks are pattern recognition systems that can be trained to recognize patterns in a noisy input environment. The architecture of these systems is frequently biologically inspired but not all neural network architectures and processes are biologically plausible. In contrast with expert systems, which are symbol processors, neural networks can be called subsymbolic since they operate at the signal processing level. Connectionism, parallel distributed computation and neural computation are other terms that are sometimes used for this field.

Neural networks are usually organized into layers of neurons. In feed-forward networks each neuron accepts input only from neurons in the preceding layer. We say that a network is fully connected if each neuron has inputs from all units in the preceding layer. See example network below.   Input signals are applied to the input layer on the left and signal activation flows from left to right. Each of the connections has an associated connection strength called a weight.

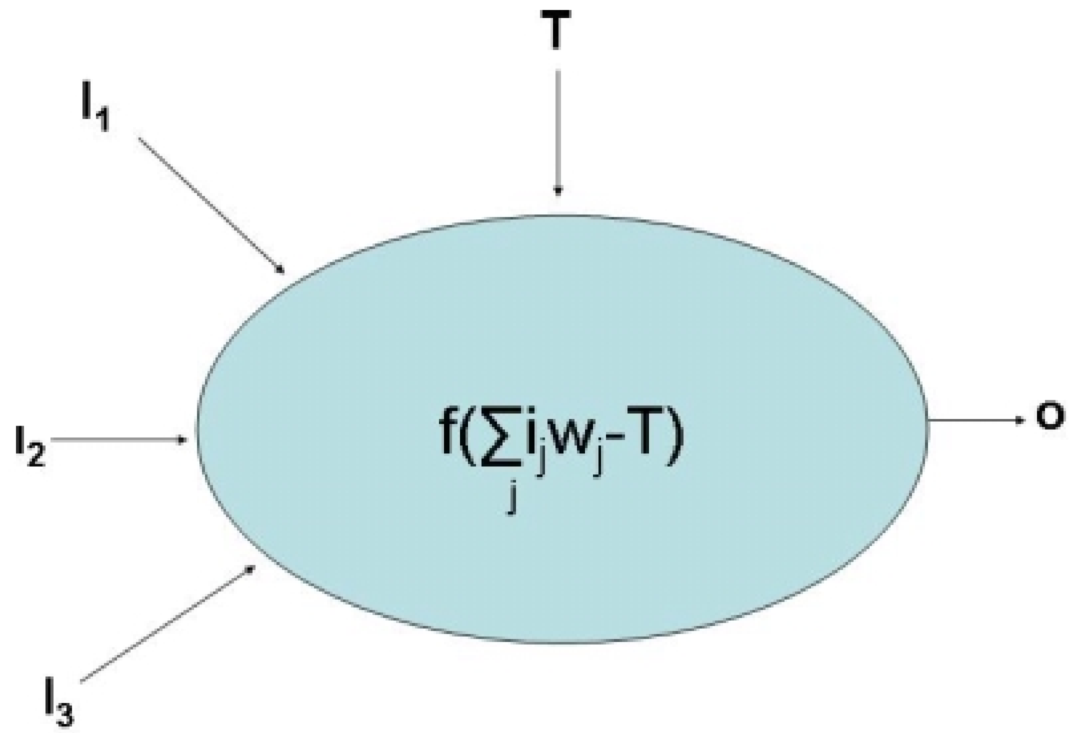


Feed-forward Neural Network

Feed-forward networks are also sometimes called perceptrons, a name given to this architecture in the 1950's. There are also more complex architectures called recurrent networks in which feedback connections are present.

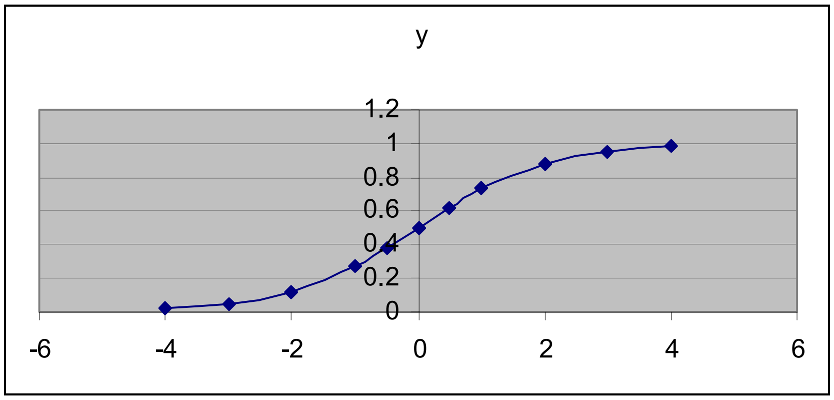
In feed-forward networks, patterns are learned through an iterative training process of adjusting the weights of the network for each pattern in a training set of examples applied to the input layer. Each iteration in the training process is called an epoch. Training is complete when the process converges on a set of weights which gives an acceptably low error on the output for all elements in the training set. In this way the network actually learns the statistics of the training set and is capable of generalization, the ability to correctly process patterns not in the training set, when in actual use.

Each neuron (sometimes called a processing element, PE), in the network is a very simple processor indeed. See the example below.



Neuron

It is basically a dot product of the input signal and weight vectors with an activation threshold T and output limiter f. Notice that a weight w_j can have an inhibitory or excitatory effect depending on whether it is adjusted so that the product a_jw_j is a high positive or negative value, respectively. Also note that if any input connection a_jw_j has a very small value relative to all other input connections, it will have little or no effect on the activation level of the neuron. The output limiter function f is often chosen to be the sigmoid function because it contributes some highly desirable computational properties during training. See the graph below. It is monotonically increasing, has limiting values of 0 and 1 and 1 and has derivative $f' = f(1 - f)$.



Sigmoid Function

A typical sigmoid function is:

$$f(a) = \frac{1}{1 + e^{-a}}$$

The training process for a feed-forward network is usually some variety of what is called supervisory training, a process in which an external “teacher” provides the “right answer” for each input pattern in the training set. There is also unsupervised training in which no teacher is provided and internal control and local information provides results such as clustering. Many supervised learning algorithms are available, the most popular being backpropagation.

Backpropagation is a form of error-correcting learning in which the weights are adjusted in proportion to the output error. The output error from a single node in the output layer is

$$\varepsilon = d - o \quad ,$$

where o is the calculated output and d is the desired output.

The weight change for each weight for a single output neuron is given by the following:

$$\Delta w = \eta \, i \, \delta ,$$

where η is a very small number called the learning rate to be discussed later, i is the input activation for that connection and δ is the error gradient at that neuron. The error gradient is defined as the derivative of the output limiter function (Recall that $f' = f(1 - f)$) multiplied by the error at the output of the neuron.

$$\delta = f(1 - f)\varepsilon$$

If $\varepsilon = d - o$ is positive, then the neuron output o is too low, so the weight is increased for positive inputs and decreased for negative inputs. For negative error values, the opposite action is taken.

The weight change calculation for each weight in the hidden layer has the same form as the output layer

$$\Delta w = \eta \, i \, \delta ,$$

except that the error ε in the error gradient $\delta = f(1 - f)\varepsilon$ term must be calculated differently since the actual network error appears only at the output layer. What is needed is an error value at the output of each hidden layer neuron so that hidden layer neurons can be trained in the same manner as the output neurons. This error value is constructed from the errors "propagated" back from the output layer to the hidden layer unit. Hence the name backpropagation. By the way, this reverse propagation of a signal is not biologically plausible. Thus for the case of a hidden layer neuron, the value of ε is

$$\varepsilon = \sum \delta w ,$$

where the sum is over all the connections to the hidden layer neuron from all output neurons.

The backpropagation training process, as with most neural network training algorithms, is an optimization search in weight space for an error minimum. The derivative comes into play since we want the weight changes to be in the direction of the greatest rate of change of the error with respect to the weights, a process called gradient descent. The learning rate η is a very small number, typically in the range of 0.001 to 0.05 to reduce the weight changes in each epoch to small values since the weight values sought are those which work for all examples in the training set.

One of the problems with neural networks is the lack of explanatory capability. This deficiency makes selling a client or management on a proposed application more challenging than expert systems. Neural networks also have problems of long training times and local minima stalls. Many enhancements have been made to the basic backpropagation algorithm to solve these problems. Some examples are as follows:

- dynamically adjusting the learning rate during training
- addition of a momentum term
- localization of learning rate to individual neurons

Given the training time problem, one might inquire about the use of a feedforward network with no hidden layer. One layer perceptrons are limited in the type of functions it can learn. With one hidden layer any continuous function can be represented and with two hidden layers discontinuous functions can be represented.

Neural networks have been used to solve prediction, clustering, classification problems.

Neural Network Resources

International Neural Network Society
http://www.inns.org/

IEEE Transactions on Neural Networks
http://www.ieee-nns.org/

Connection Science
http://www.ingenta.com/journals/browse/tandf/ccos

Cognitive Science
http://www.sciencedirect.com/cogsci

Module #4: Genetic Algorithms

Definition: A genetic algorithm (GA) is a problem solving approach inspired by biology in which the space of potential problem solutions is searched.

GA's differ from traditional optimization and search procedures in four ways:

1. GA's operate on a coded representation of the parameter set and not directly on the parameters themselves.

2. GA's are searching the space with a population of points

3. GA's use payoff (objective function) information and not derivatives

4. GA's use probabilistic transition rules instead of deterministic rules

The population of candidate solutions is represented as chromosomes that are sequences of genes each of which has a value. Successive generations of these chromosomes are generated with reproduction for a given generation being a function of chromosome fitness with the less fit not being passed to the next generation. GA's require these chromosomes to be coded as finite strings which are often binary. The basic operations are reproduction, crossover and mutation.

The basic genetic algorithm process can be described as follows:

1. Problem description and design of solution representations

2. Randomly generate an initial population of candidate solutions

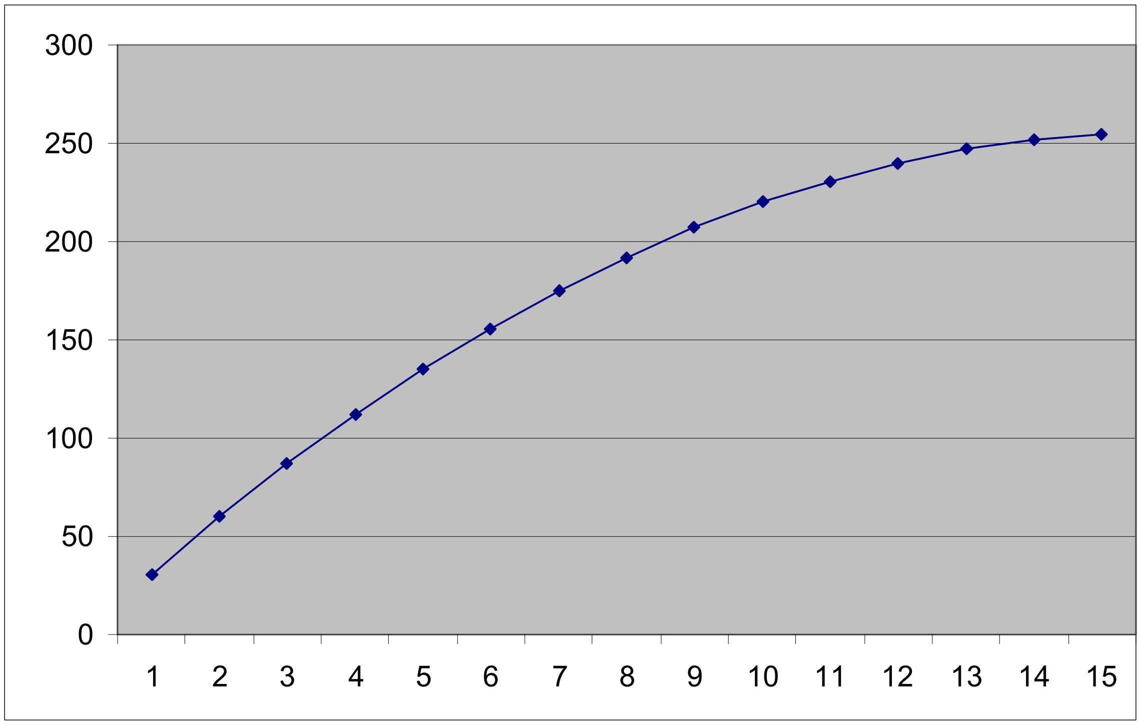
3. If a solution is good enough then stop.

4. Select the best candidate solutions from the population as parents for producing the next generation.

5. Apply crossover operations and create the next generation

6. Go to step 3

We illustrate the algorithm with the problem of finding the maximum value of the function $g(x) = 32x - x^2$ over the interval [1,16] assuming a domain of integer values for x.

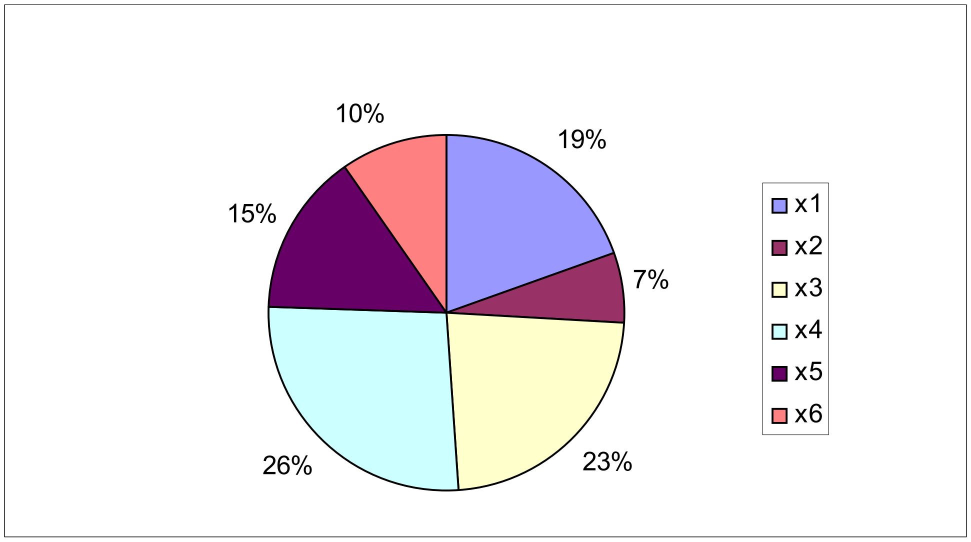


Graph of $g(x) = 32x - x^2$

We choose a population size of six and randomly generate the following initial population.

Label	Chromosome string	String value	String fitness	Fitness Ratio
X1	00111	7	175	19%
X2	00010	2	60	7
X3	01001	9	207	23
X4	01100	12	240	26
X5	00101	5	135	15
X6	00011	3	87	10
			-------	------
			904	100

The string fitness is the value of g(x). Since we are interested in the maximum value of g, we want the reproduction operator to favor the higher fitness values as candidates for producing the next generation. This can be done by using fitness ratios. The fitness ratios are computed by dividing each string fitness value by 904, the fitness total of all six strings. These values can be used to bias the reproduction operator toward generating higher fitness strings for the mating pool for the next generation using a roulette wheel in which the slot sizes correspond to each string and its associated fitness value. Execution of the reproduction operator corresponds to the spinning of this weighted roulette wheel, generating a string six times to produce the next generation mating pool, with each spin producing the string associated with the slot at which it stops. Notice that this procedure produces a new set of strings with the intended fitness bias.



Weighted Roulette Wheel

In our example, string x4 has a fitness value of 240 which represents 26% percent of the total population fitness. Therefore, string x4 is given roulette wheel weight 26. Each spin of the wheel gives string x4 with probably 0.26.

We now have six strings which are copies produced from the initial population by a procedure designed to increase the average fitness. We now randomly pair these strings for a process called crossover. Crossover involves string segment swapping between the pairs. We can illustrate this process with the following example:

Suppose we have two strings.

11111

and

10000

The segment swapped is defined by a location on the string. Suppose this point is between the third and fourth character in the string. Then the result of the crossover would be the following two strings:

11100

and

10011

The crossover point is randomly selected for each string pair. After this crossover (mating) operation is carried out for each pair, we have six new strings which represent the next generation population. In some cases a mutation operation is also applied at this stage. Mutation means the random selection and change of a single bit position. Its purpose is to help assure that the search process does not get stuck on a local optimum. We can now assess the fitness of each string in this generation as we did for the initial population. This iterative process is continued with average fitness improving each generation until the population produces a near-optimal solution. Hundreds of generations may be required.

GA's have been used widely in optimization problems such as circuit layout and scheduling, but they are computationally intensive and the conditions under which they perform well remain research issues. The design of appropriate chromosome representation in complex problems is crucial to success and requires careful engineering.

Genetic programming is closely related to GA's. The primary difference lies with the representations that are combined and mutated. In genetic programming these structures are programs rather than bit strings. Program representations are expression trees that can be in a language such as Lisp or they can be designed to represent objects in the problem domain. As with GA's, there are questions about the effectiveness of genetic programming. The term evolutionary computation is being used to include the combined fields of GA's and genetic programming.

Genetic Algorithm Resources

Genetic and Evolutionary Computation Conference
http://www-illigal.ge.uiuc.edu:8080/GECCO-2004/, a recent merger of the earlier International Conference on Genetic Algorithms and the Conference on Genetic Programming.

Evolutionary Computation
http://www.aic.nrl.navy.mil/~aswu/ecj/

IEEE Transactions on Evolutionary Computation
http://www.ieee.org/portal/index.jsp?pageID=corp_level1&path=pubs/transactions&file=tec.xml&xsl=generic.xsl

Recommended Books

Mitchell, Melanie., **An Introduction to Genetic Algorithms**, MIT Press, Cambridge, MA, 1996.

Fogel, David, **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**, IEEE Press, Piscataway, N.J., 2000.

Module #5: Fuzzy Logic

One might say that "traditional" numerical and even symbolic computing has a precision and brittleness that fails to capture some of the fundamental aspects of human problem solving processes. For example, expert problem solvers are quite comfortable dealing with imprecise "soft" terms such as "low temperature" and "high pressure". Since this type of imprecision is clearly part of the problem solving process of experts, it would be desirable to incorporate this capability into AI methods directly. Fuzzy logic was developed for just that purpose. Fuzzy logic deals with degrees of set membership and degrees of truth.

In classical set theory, a set A is defined in terms of elements of a universe U being either a member of A or not a member of A. We could also express this definition in terms of a set membership function m which maps the elements of U onto the two element set {1,0}. That is,

For all elements in U

m(x) = 1 if x is in A and

m(x) = 0 if x is not in A

In a similar fashion, we can define fuzzy sets. A fuzzy set A of elements from a universe U is defined by a membership function m that maps the elements of U into the interval [0,1] of real numbers. We can express this definition as follows:

For all elements x in U

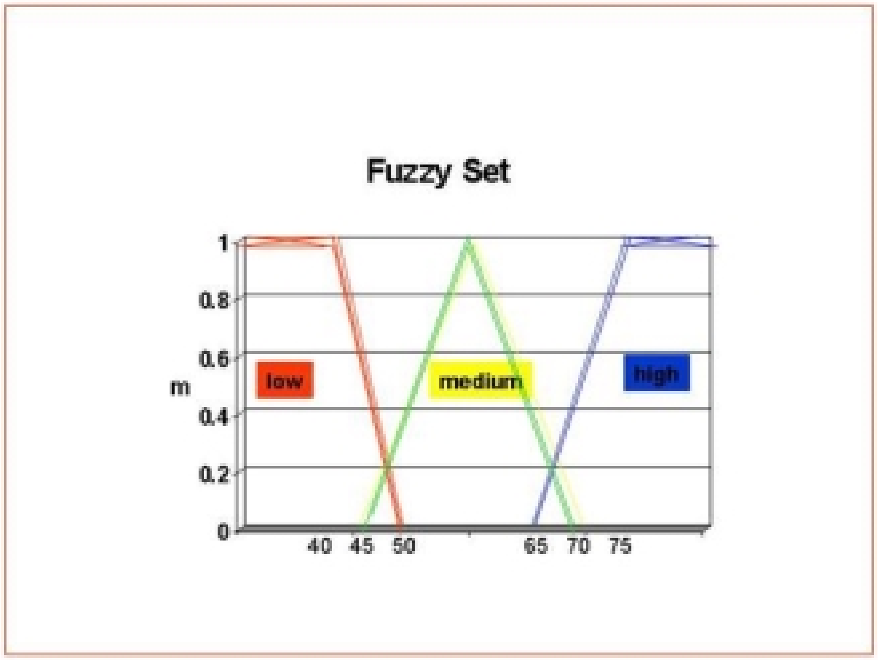
m(x) = 1 if x is completely in A

m(x) = 0 if no part of x is in A

$0 < m(x) < 1$ if x is partially in A

For fuzzy sets, m is said to represent the degree of membership of x in A. Sets with membership function values of only 0 and 1 are said to be crisp sets in fuzzy set lingo.

Fuzzy set membership functions could be represented many ways mathematically. For practical computational efficiency purposes a popular representation is the piecewise linear function. We illustrate with the example of high speed and low speed in the graph below.



Fuzzy sets are also often represented as fit vectors.

High speed = (0.0/40, 0.0/45, 0.0/50, 0.0/57.5, 0.0/65, 0.5/70, 1.0/75)

Medium speed = (0.0/40, 0.0/45, 0.4/50, 1.0/57.5, 0.4/65, 0.0/70, 0.0/75)

Low speed = (1.0/40, 0.5/45, 0.0/50, 0.0/57.5, 0.0/65, 0.0/70, 0.0/75)

We can define fuzzy set operations analogous to crisp sets. Let A and B be fuzzy sets over some universe U with membership functions $m_A(x)$ and $m_B(x)$ respectively. The fuzzy set union of A and B, is defined as the set AUB with membership function

$$m_{AUB}(x) = \max\{m_A(x), m_B(x)\}, \text{ where } x \varepsilon U.$$

Similarly the intersection of fuzzy sets A and B is defined as A∩B with membership function

$$m_{A\cap B}(x) = \min\{m_A(x), m_B(x)\}, \text{ where } x \varepsilon U.$$

The fuzzy set complement of A is defined as $A^{'}$ with membership function

$m_{A'}(x) = 1 - m_A(x)$, for $x \varepsilon U$.

We illustrate these operations with the example speed and temperature fuzzy sets above.

High speed OR medium speed = (0.0/40, 0.0/45, 0.4/50, 1.0/57.5, 0.4/65, 0.5/70, 1.0/75)

Fuzzy set intersection

Low AND medium speed = (0.0/40, 0.0/45, 0.0/50, 0.0/57.5, 0.0/65, 0.0/70, 0.0/75)

Fuzzy set complement

NOT high speed = (1.0/40, 1.0/45, 1.0/50, 1.0/57.5, 1.0/65, 0.5/70, 0.0/75)

Fuzzy logic can be implemented in rule-based systems by providing support for rule conditions and conclusions that are fuzzy sets. Fuzzy reasoning provides for more flexibility and may allow for several options rather than a single conclusion from the system, providing the basis for better decision-making judgment. In crisp rule based systems, the rule condition is either true or false, and if true, the rule conclusion is taken to be true. In fuzzy rule-based systems rules can fire partially. The rule condition describes the degree to which the rule applies and the rule conclusion assigns a membership function to each of the output variables. If the condition is true to some degree of membership, then the conclusion is also true to the same degree. The condition part and conclusion part can contain Boolean combinations. For example,

IF speed is medium

AND pressure is high

THEN risk is high

AND system is abnormal

The inferencing process involves several steps:

- mapping crisp inputs into the rule conditions,

- partial firing of rules which have nonzero condition membership function values under this mapping,

- combination of output variable membership functions and finally

- defuzzification of this resulting combination.

The result of final step of defuzzification is a crisp output that is a centroid value or weighted average value computation, depending on whether the computer representation of the membership function is the graph shown above (Mamdani-syle inference) or a more computationally efficient internal representation (Sugeno-style inference), respectively. (See following website for more details. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/faqs/ai/fuzzy/part1) The MATLAB Fuzzy Logic Toolbox (http://www.mathworks.com/products/fuzzylogic/) is one of the most popular tools for fuzzy expert system building.

The use of fuzzy rules can substantially reduce the total number of rules in an expert system. However, a downside is that the resulting knowledge base requires a time consuming stage of performance tuning which involves adjusting the fuzzy sets and rules.

Fuzzy Logic Resources

IEEE Transactions on Fuzzy Systems
http://ieee-cis.org/pubs/fts

International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems
http://www.worldscinet.com/ijufks/ijufks.shtml

Module #6: Emerging AI Technologies

Soft Computing

Soft computing is a label that can be given to a computational technique that is tolerant of imprecision and uncertainty. Therefore, fuzzy logic, neural networks, genetic algorithms and probabilistic reasoning in expert systems that have been discussed in the previous sections certainly qualify. However, it is important to recognize that soft computing is not simply a label for a collection of AI technologies but is viewed as an emerging new paradigm within which existing techniques can be combined to form more effective problem solving approaches. See website http://www.soft-computing.de/def.html.

Soft Computing Resources

Journal of Multiple-Valued Logic and Soft Computing
http://www.oldcitypublishing.com/MVLSC/MVLSC.html

Applied Soft Computing Journal
http://www.ingenta.com/journals/browse/els/15684946

World Federation of Soft Computing
http://www.softcomputing.org/

Applied Soft Computing
http://www.eeaax.polytechnique.fr/marc/ASC.html

Berkeley Initiative in Soft Computing
http://www-bisc.cs.berkeley.edu/

Chat Bots

Chatbots (frequently called bots, and sometimes called v-people or synthetic characters) represent an emerging technology of human-computer interaction. They basically implement a natural language interface which is designed to simulate conversation by using a variety of methods. Recent advances in computer graphics have made it possible to embody these bots as realistic "synthetic characters", not only for interactive games, but for providing Internet information services, training and web site guides. In response to user input questions or statements, the bot consults its knowledge base and formulates a reply. These replies are functions of pattern matching algorithms which recognize key words, phrases and sometimes sentences. Some sophisticated bots employ learning algorithms that allow the bot to remember input sequences of information for later responses and hence expand its knowledge base for future interaction sessions.

There is evidence that attractive bots on web sites can significantly increase the site hit rate. Therefore, it is not surprising that chat bot technology is beginning to be commercialized with the emergence of bot customizing and hosting services for commercial websites. Research on the benefits of interactive online characters makes the case for the importance of what is called "social intelligence", exhibited as facial and emotional expressions, gestures and speech and language abilities in human-computer interactions, citing the following ten benefits of synthetic online characters - Byron Reeves paper: http://www.stanford.edu/~reeves/

- Characters make explicit the social responses that are inevitable.
- Interactive characters are perceived as real social actors.
- Interactivity increases the perceived realism and effectiveness of characters.
- Interactive characters increase trust in information sources.
- Characters have personalities that can represent brands.
- Characters can communicate social roles.
- Characters can effectively express and regulate emotions.
- Characters can effectively display important social manners.
- Characters can make interfaces easier to use.
- Characters are well liked.

Interest in conversational interfaces dates back to the mid twentieth century. The famous Turing Test for machine intelligence was proposed as standard for assessing whether a computer system had attained intelligence equivalent to humans. In this test the computer system and a human are available for communication in separate rooms. Which room contains which is not revealed to the person allowed to communicate. If after some period of communication the person communicating believes the room containing the computer is a human being, the test would be passed. To encourage the development of conversational interfaces, the Loebner Prize has recently been established and an annual competition based on the Turing Test is held. (See web sites http://www.loebner.net/Prizef/loebner-prize.html and http://www.alicebot.org/). No system has yet won the Prize, but annual awards are given to the best system each year as ranked in several categories. A bot named ALICE built using the Artificial Intelligence Mark Up Language (AIML) has won the annual award. ALICEbot technology is today's version of a famous natural language interface called ELIZA built in 1966 that was designed to simulate a psychotherapist. It was a rather simple conversational interface that responded to users by asking leading questions triggered by nouns in the user input. For example, if the user mentioned something about their mother, the system might respond with "Tell me more about your mother.". ELIZA's developers, Weizenbaum and Colby, were very surprised at the high level of intelligence that users attributed to the system as indicated by the personal revelations these sessions elicited. However, they did not see this surprising user reaction as a reason for celebration but rather distain because of the simplicity of the system. See web sites http://www.linux-mag.com/1999-06/perl_01.html and http://www.eliza.

The recent resurgence of interest in chat bots has produced a variety of tools and resources for building these bots. Some examples are listed below.

Chat Bot Resources

Oddcast: Markets conversational character products
http://www.oddcast.com

Extempo Systems: Develops synthetic character technologies
http://www.extempo.com

A.L.I.C.E. Foundation: Source of public domain tools and information about Alicebots
http://www.alicefoundation.org

Speech Synthesis Markup Language (SSML)
http://www.w3.org/speech-synthesis/

Artificial Intelligence Markup Language (AIML)
http://www.aiml.org

Microsoft Agent Technology
http://www.msagentring.org

Prendinger, Helmut and Mitsuru Ishizuka (Editors), **Life-Like Characters: Tools, Affective Functions, and Applications**, Springer, 2004.

Plantec, Peter, **Virtual Humans: Creating the Illusion of Personality**, American Management Association, 2004. (Includes CD with software tools).

Leonard, Andrew, **BOTS: The Origin of New Species**, Hardwired, San Francisco, 1997. (A history of the bot-creating culture)

Course Summary

Artificial intelligence technologies have found increasing engineering applications in real-life industrial situations. Many organizations have been successful in using intelligent systems in their day to day operations. This course has presented a basic introduction to four of the most popular artificial intelligence techniques of expert systems, neural networks, genetic algorithms and fuzzy logic together with a discussion of the emerging AI technologies of soft computing and chat bot agents.

General AI Resources

American Association for Artificial Intelligence (AAAI)
http://www.aaai.org

International Journal of Human-Machine Studies
gort.ucsd.edu/newjour/i/msg02351.html

IEEE Transactions on Intelligent Systems
http://www.computer.org/intelligent/

AI Magazine
http://www.aaai.org/Magazine/magazine.html

Applied Artificial Intelligence Journal
http://www.tandf.co.uk/journal/titles/08839514.as

Negnevitsky, Michael, **Artificial Intelligence: A Guide to Intelligent Systems**, Addison-Wesley, Pearson Education Limited, Essex, England, 2005.

Russell, Stuart and Peter Norvig, **Artificial Intelligence: A Modern Approach**, Prentice Hall, Second Edition, 2003.

Resources for Engineering Applications of AI

Artificial Intelligence in Engineering, An International Journal,
http://www.ingenta.com/journals/browse/els/09541810

Engineering Applications of Artificial Intelligence, The International Journal of Intelligent Real-Time Automation.
http://www.elsevier.com/wps/find/journaldescription.cws_home/975.........

AIEDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing, a journal carrying articles of both theoretical and practical interest.
http://www.cs.wpi.edu/~aiedam/index.html

Artificial Neural Networks In Engineering, (ANNIE) conference
http://web.umr.edu/~annie/

International Conference on the Applications of Artificial Intelligence to Civil and Structural Engineering, http://mason.gmu.edu/~tarcisze/conferences..htm

Hopgood, Adrian A., "Knowledge-Based Systems for Engineers and Scientists", **CRC Press**, Boca Raton, Florida, 1993.

Loi Lei Lai, **Intelligent System Applications in Power Engineering: Evolutionary Programming and Neural Networks**, John Wiley & Sons, 1998.

Thomas Quantrille and Y. Liu, **Artificial Intelligence in Chemical Engineering**, Elsevier, 1992.

H.E. Rauch (Editor) "Artificial Intelligence in Real-Time Control", **Proceedings of the IFAC Symposium**, Kuala Lumpur, Malaysia, September 1997, Elsevier.