



SOFT COMPUTING

Angela Bower

TESSELLA SUPPORT SERVICES PLC

Issue V1.R1.M0

July 2003



Soft computing is an umbrella term for a collection of computing techniques. The term was first coined by Professor Lotfi Zadeh, who developed the concept of fuzzy logic in the mid 60's. In his words:

“Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty and partial truth. In effect, the role model for soft computing is the human mind. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness and low solution cost”

The main techniques in soft computing are evolutionary computing, artificial neural networks, fuzzy logic and Bayesian statistics. Each technique can be used separately, but a powerful advantage of soft computing is the complementary nature of the techniques. Used together they can produce solutions to problems that are too complex or inherently noisy to tackle with conventional mathematical methods.

Evolutionary Computing

Evolutionary computing uses the Darwinian principle of ‘survival of the fittest’ to evolve optimum solutions to problems.

There are a number of evolutionary techniques whose main similarity is the use of a population of random or pseudo-randomly generated solutions to a problem, each of which is judged on their adaptation to the problem being addressed using a ‘fitness function’.

The fitness function evaluates the solution and returns a numerical answer, which rates the solution's suitability. This population is then used to generate a new population of solutions, by using one or more of the following genetic operators:

- Re-combination/cross-over: This operation involves combining the ‘genetic’ information from two parent solutions to produce an offspring solution
- Mutation: This operation involves taking a solution and mutating its genetic information slightly
- Cloning: This operation copies the solution to produce a clone.

The solutions that are used in these operations are selected from the main population pool and placed into the ‘breeding pool’ using a selection method. The protocol for creating this breeding pool differs between the various algorithms, as does the protocol for deciding which candidates enter the next generation. All the techniques require the population size to remain constant between generations.

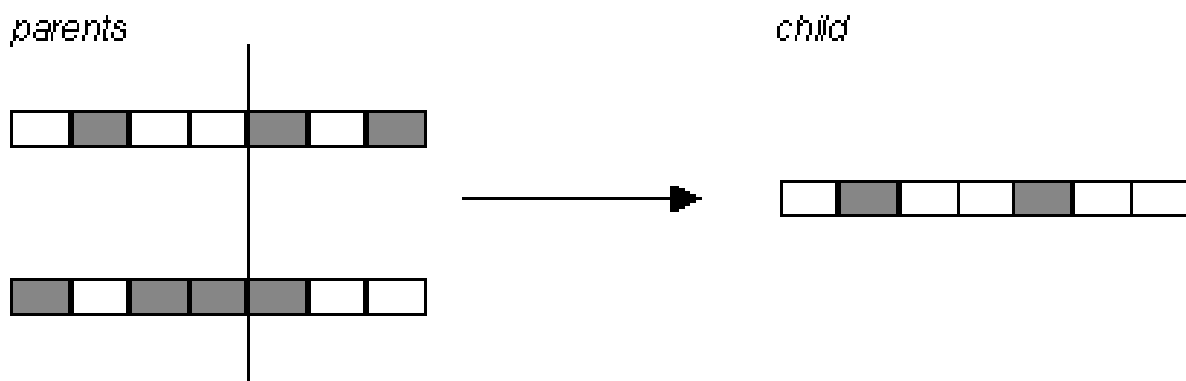
This evaluation and regeneration of the candidate populations is repeated until a near-optimal solution has been created.

Genetic Algorithms

Of all the evolutionary techniques genetic algorithms have proved to be the most commercially successful.

The solutions in these algorithms are encoded into strings. The strings represent the properties that the resulting solution will have, for example a list of the values of variables in an equation or the properties of components in a design.

New generations are created using all the genetic operators, but the recombination operator is given a much higher priority than the mutation and cloning operators, and mutation is usually only used as a ‘background’ operation to maintain diversity in the population. Re-combination is achieved by combining sections of the encoded strings from both parents to produce a child, as shown below:



The mutation operator usually performs point mutations. Where values are encoded in binary, this can be as simple as using a bit-flip to alter the values, but where real numbers are used, a random number can be added or subtracted instead.

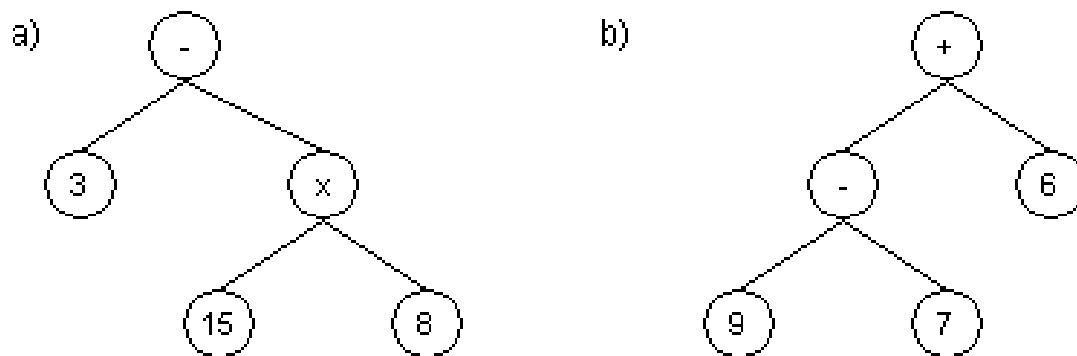
The parent solutions that take part in creating the new generation are chosen based on their fitness. The most common way to do this is to evaluate the whole population using the fitness function and then produce a breeding pool using fitness proportional representation, so that the most successful candidates are more frequently represented. Another way involves taking random groups of candidates from the main pool and competing them to find the fittest candidates in each group. Both of these techniques result in a selected pool that has a high proportion of 'good' solutions, but which does not exclude the 'bad' solutions. This is important to balance the need to create an optimum solution against the possibility of becoming too set on one evolutionary path, which may not lead to the overall optimum solution.

The replacement policy that is usually applied involves replacing the whole of the previous generation with the newly created individuals, but more subtle methods can be used where the generated children only replace the weakest individuals in the parent population.

Genetic Programming

Genetic programming works in a very similar way to genetic algorithms. The main difference between the two techniques is the representation used for the candidate solutions.

In genetic programming, the solutions are actual computer programs. As a result the representation of the candidate's genetic material needs to be more complex. Most commonly a tree structure is used to represent the structure of the program. The leaf nodes in this structure are values, and the internal nodes are functions selected from an allowed set. These functions typically would include arithmetic and Boolean operators as well as conditional functions.

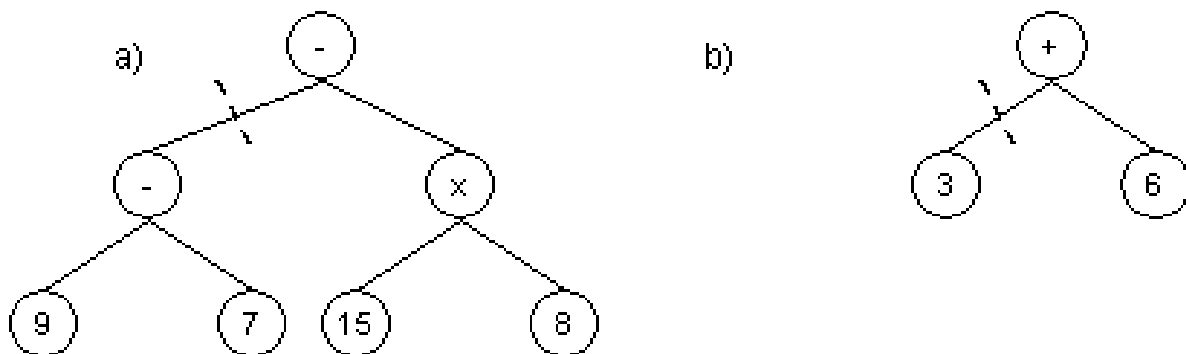


Evaluates to:

a) $3 - (15 \times 8)$

b) $(9 - 7) + 6$

The representation of genetic programs requires the genetic operators to act in a slightly different way. The re-combination operator is still the most dominant, but its actions are altered to deal with tree structures. Usually re-combination involves selecting a sub-tree at random from each parent, and then swapping them.



Evaluates to:

a) $(9 - 7) - (15 \times 8)$

b) $3 + 6$

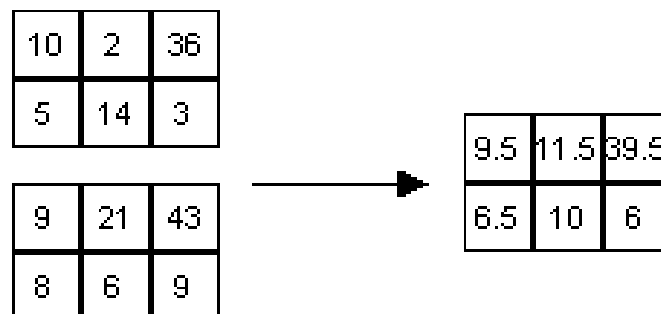
Similarly the mutation operator replaces a randomly selected sub tree with a randomly generated sub-tree.

Evolution Strategies

Evolution strategies were developed by researchers at the Technical Institute of Berlin, who wanted to automate the design of aerodynamic jointed plates. The representation that they used for the candidate solutions was a list of fixed-length, real-valued vectors, which describe the characteristics of the design.

Initially they took a single parent design, mutated it and then applied the fitness function to both the child and the parent, choosing the fittest individual to carry on to the next generation. This approach has since been expanded to allow populations of parents and children. The parents are selected entirely randomly and are used to produce a child population with a size that is greater than the population size.?

Originally the only genetic operator used was mutation, but an intermediate re-combination operator can now be used to blend the characteristics of the parents. This operator works on the values in the parent vectors, and averages the values to produce the child. However, unlike genetic algorithms, mutation is the primary operator, with re-combination only used secondarily.



Recombination in evolution strategies

There are two main types of replacement policies used, namely (μ, λ) and $(\mu + \lambda)$. In this notation μ is the population size and λ is the number of children produced. In the comma method the next generation is selected from the children, using the fitness function to determine the fittest solutions. The plus method selects best candidates from the parents and the children to make up the next generation.

Evolutionary Programming

The final technique is often grouped together with evolution strategies, as they are very similar in their approach. Both tend to use fixed-length real-valued vectors to represent the solutions and both use the fitness function to select which candidates enter the next generation rather than which candidates become parents.

In evolutionary programming the only genetic operator used is mutation, and it is used on the whole of the current generation to produce an equal number of children. The next generation is then selected from this combined pool using the fitness function to choose the best candidates.

Uses of Evolutionary Computing

Evolutionary computing has already proved itself as a useful tool in a range of industries. One example of this is the pharmaceutical industry, where genetic algorithms have been applied to computer-aided molecular design (CAMD).

Designing a molecule to order is a difficult task. To find a new drug to bind to a known receptor you first need to analyse the structure of either other known ligands or the receptor itself to produce information on the functional areas of the molecules involved.

This information can be used to assess the binding properties of new molecules, but the new molecules still need to be designed. Due to the complexity of the search space, conventional rule-based or numerical methods tend to be unsuccessful.

This is where genetic algorithms can help. By encoding the structure of the chemicals into chromosomes or graphs, genetic algorithms can be used to evolve new chemicals. The fitness function must then build the molecules, checking that they make chemical sense, before assessing their binding properties. By using evolution unexpected chemicals can be produced, which can give chemists insights and suggestions into potential chemical families to investigate.

Fuzzy Logic

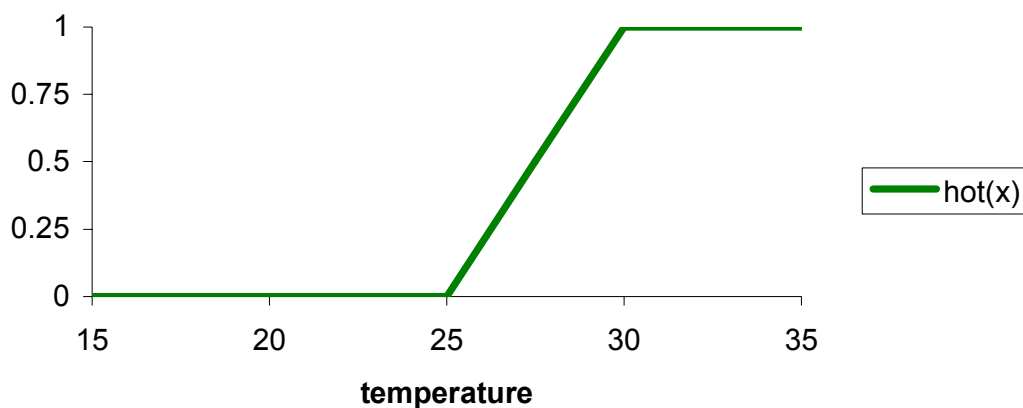
In Boolean logic, the only values allowed are completely true and completely false. This black and white representation of the world can be difficult to implement. If, for example, if you had a group of air temperatures and you asked

the question ‘is x hot?’ then it would be difficult in many cases to give a definitive yes or no answer. One way to get around this in Boolean logic would be to define a strict boundary and say that any temperature over 30°C is hot, but this is very artificial. Why should 30.1°C be hot when 30°C is not?

In fuzzy logic you can assign degrees of truth and falsehood. If we use the example above, a temperature of 27°C would be considered ‘fairly’ hot, whereas a temperature of 25 would be ‘not very’ hot. These answers are given a numeric value in the interval 1 to 0, where 1 is the classical Boolean value true and 0 is false, for example ‘fairly’ could be translated to a value of 0.7 and ‘not very’ as 0.1.

Fuzzy Sets

This leads us into the idea of fuzzy sets. If we keep the example above, the question ‘is x hot?’ is a way of defining a subset of hot temperatures. In Boolean logic the members of the super set (called the ‘universe of discourse’) either belong to the hot subset or do not belong. With fuzzy sets, individuals can be given a degree of membership to a subset. This degree of membership is calculated using a membership function. An example of a membership function for the fuzzy subset ‘hot’ could have the distribution shown below:



This is a fairly simple membership function, but more complex functions can be used.

Fuzzy Logic operators

Once you have converted all your inputs into fuzzy values, a process known as fuzzification, you can now apply fuzzy logic. In Boolean logic algebraic sentences are created using the operators AND, OR and NOT. These operators can also be used in fuzzy logic. The most common interpretation of these operators in fuzzy logic is:

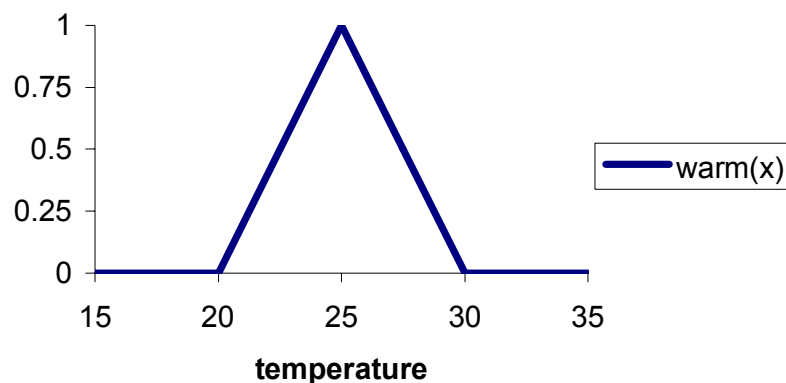
$$\text{NOT } x = 1-x$$

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

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

These operators produce the same results for the true and false values (1 and 0) as Boolean operators, which shows that Boolean logic is simply a subset of Fuzzy logic.

For example, if we define a new fuzzy set for warm temperatures. The membership function could look as follows:



We can now construct some logical expressions to show how the fuzzy logic operators work. If a member of the universe of discourse were a temperature of 26.5°C, the membership functions for the two sets would give values of .3 for the hot set and 0.7 for the warm set. We can now ask questions such as

Is x NOT hot?

Which would give the result: $1 - 0.3 = 0.7$

Is x hot AND warm?

Which would give the result: $\text{MIN}(0.3, 0.7) = 0.3$

Is x hot OR warm?

Which would give the result: $\text{MAX}(0.3, 0.7) = 0.7$

Fuzzy Inference Systems

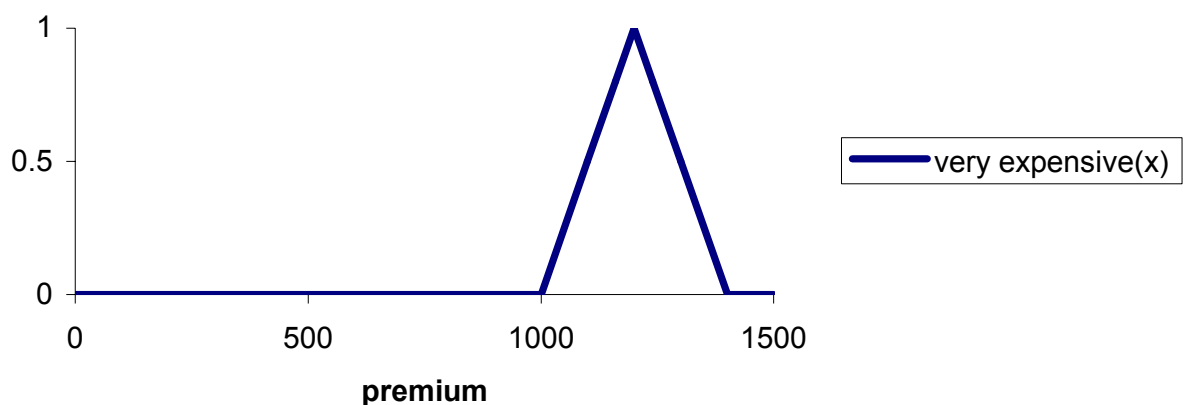
A fuzzy inference system uses fuzzy logical rules to produce output values from input values. These rules take the form:

(antecedent) IF car is powerful AND driver is young

(consequent) THEN insurance premium is very expensive

The antecedent will produce a numerical answer, for example a 20 year old driver in a Ferrari could be given membership values of 0.8 for young and 0.9 for powerful giving the antecedent an answer of 0.8.

The consequent assigns a fuzzy set to an output variable, using the answer to the antecedent to alter the fuzzy set's distribution. For example in the rule above we are setting the value of the output variable 'insurance premium' to the fuzzy set 'very expensive', which would have the distribution shown below:



The degree to which the output variable belongs to that fuzzy set is defined by

the antecedent, so to combine the two parts we truncate the output fuzzy set at the value returned. This is called implication. In this example the antecedent returned a value of 0.8, so we ‘cut’ the top of the output fuzzy set off at that value.

This fuzzy set output now needs to be converted into a crisp output value in order to be of any use, a process called de-fuzzification. The most conventional way of doing this is to find the centroid of the fuzzy set’s distribution, which is the point at the centre of the area under the curve.

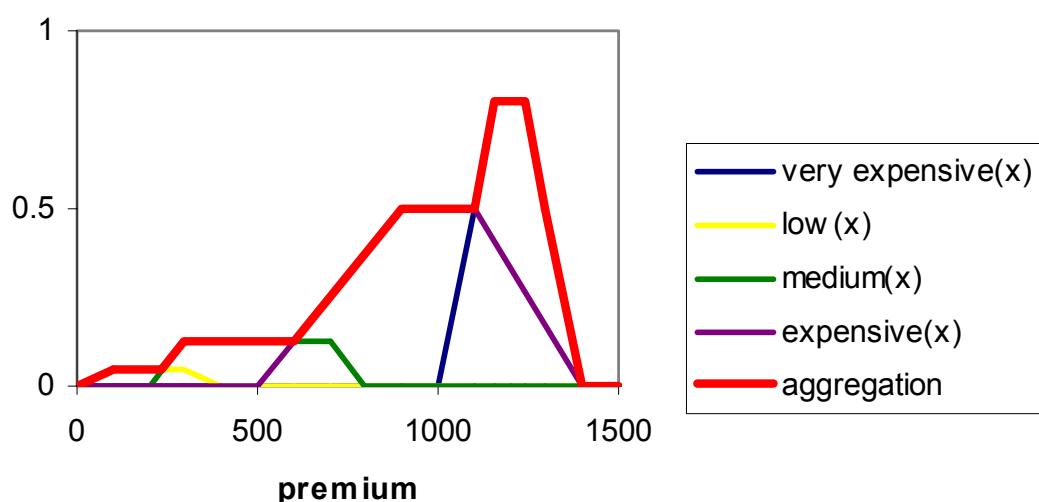
When more than one rule is defined all the rules are evaluated in parallel. This leads to a situation where the output values can have more than one fuzzy set assigned to them. These fuzzy sets first need to be aggregated before they can be de-fuzzified, which is done usually done by applying a MAX operator to them. For example if the insurance premium inference system had three more rules:

IF driver is mature and car is average THEN premium is medium

IF no claims period is long THEN premium is low

IF no claims period is short THEN premium is expensive

The resulting premium before de-fuzzification could look like this:



This would then be de-fuzzified, using the centroid function to give a crisp output value.

Uses of Fuzzy Systems

In Japan, Fuzzy Logic has been used successfully for decades. There are a number of high profile cases that use Fuzzy Logic, for example the subway trains in the Japanese city of Sendai are controlled using a fuzzy logic system. This system was designed using train drivers expert knowledge and has been shown to accelerate and brake more smoothly than human operators and stop at the stations with more accuracy.

More recent examples include the incorporation of fuzzy logic controllers in ABS technology. Braking of a vehicle occurs when the wheels rotate slower than the speed of the vehicle, but when the wheels stop moving completely, i.e. when the car skids, it is less easy to steer and braking is not optimal. The braking effect is related to the difference in velocity between the wheels and the car, known as the 'slack'. The ABS controller adjusts the by-pass valves on the brake fluid to keep the slack on the wheels optimal for the braking effect.

Unfortunately this is not a linear problem, as the optimal level of slack differs depending on the road conditions. Most ABS systems therefore compromise and set the target level to 0.1, but if a system could be developed to assess the condition of the road then much better performance could be achieved. A number of companies have researched using sensors on the car to detect these conditions, but it has been shown that the cost of this would be prohibitive.

Fuzzy logic provides an alternative solution, by using the sensors already in place to feedback the condition of the road once braking has started. Much as the human driver can estimate the road conditions by assessing the effects of applying the brakes, the fuzzy logic controller has a rule-base that uses the speed of the car, the speed of the wheels and the pressure of the brake fluid to calculate the change that should be made to the braking profile.

Research has shown that using only six fuzzy-logic rules gives a significant improvement in braking performance. As fuzzy logic systems have a high computational efficiency, this calculation can be performed easily within the time constraints of a successful control loop. This has been shown to enable the controller to adjust to changing road conditions during braking.

The technology also lends itself to developing decision support systems for areas where expert knowledge is required. One successful example in the financial sector is a system for differentiating fraudulent insurance claims from genuine ones, helping to make the efforts of fraud investigators more targeted. A system was developed using expert knowledge from a range of insurance companies and it has been demonstrated to automatically settle up to 85% of all insurance claims.

Artificial Neural Networks

In human brains large numbers of neurons with numerous connections to one another are able to calculate highly complex problems and react to new situations by learning from previous experiences.

This is the basis of artificial neural networks (ANNs), although on a vastly more simplified level. The basic unit of an artificial neural network is an artificial neuron. It receives numerical inputs and produces an output based on these, and possibly on a small local memory store. These units are built up into networks that can learn from input data to perform complex calculations that are generalised to the problem and therefore tolerant of noisy data and able to work on previously unencountered input data.

Artificial neurons

Artificial neurons, or nodes, are built on very simple models of real neurons. In a real neuron the cell receives electrical inputs through its dendrites. These reach the cell body where they summate. If the summation of the inputs is over a threshold level for that neuron it fires off an electrical signal down its axon.

This is replicated in the artificial neuron, whose ‘dendrites’ are numerical inputs, which are summated and then subjected to a threshold function to determine the output of the unit. The threshold function can be linear or more complex, for example sigmoidal. Importantly the inputs are also given weights, to show how important they are to the function. These weights can be positive or negative and they are used when summing the inputs to determine the total level of activation of the unit.

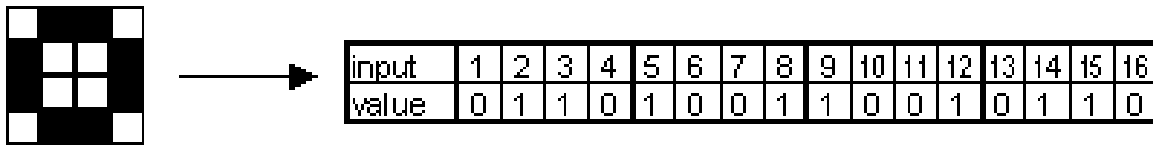
These weights and the threshold function itself can be adjusted to alter the output of the neuron. This ability allows the neuron to be trained to perform a specific function. Given a known set of data, perhaps experimentally obtained, the artificial neuron can be initialised with a number of inputs to match the number of inputs in the data, each with a random initial weight. When this neuron is

presented with the first row of data in the set, the output can be compared to the actual measured output, to find the degree of error. This figure can be fed back to slightly adjust the weights on the inputs proportionally to their current weight. This process continues until the neuron converges, which occurs when the error rate of the neuron, when classifying a complete pass through the data set, falls below a set tolerance.

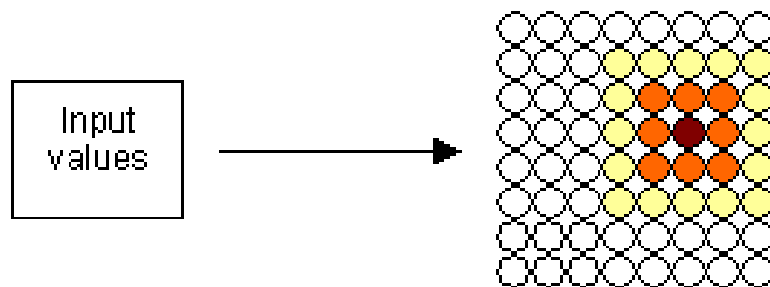
By themselves artificial neurons are only capable of calculating linearly separable problems, but when they are grouped into networks they can be trained to map any function. There are different configurations that can be used, two of which are explained below.

Self Organising Feature Maps

The first configuration is the Self Organising Feature Map (SOFM). These networks are used to find groupings within an input dataset. A 2D network of nodes is topologically mapped to the input data. Each node in the grid receives all the inputs, but all these connections are given initial random weights. When a set of data is presented to the system, each node will have a different level of activation. To train the SOFM, the 'winner' node is found, by finding the node whose input weights are closest to the input pattern. The weighting of the node's inputs are altered to make it respond to the input data set more strongly. Similarly, the connections of the surrounding neurons in the grid are strengthened in proportion to how near each node is to the winner. This is repeated with different inputs in the data set until the weight changes are nominal.



The pixelated image is converted to a binary set of inputs



The red node is the winner and its input weights and those of its neighbours are altered to increase the response to the input.

The effect of this training is to create topographical regions in the SOFM that respond strongly to certain cases of input data. For example if the SOFM were trained on pixelated alphanumeric characters, different regions would be activated when presented different characters. Characters that bear similarities to each other would have activation regions close to one another, for example O and C.

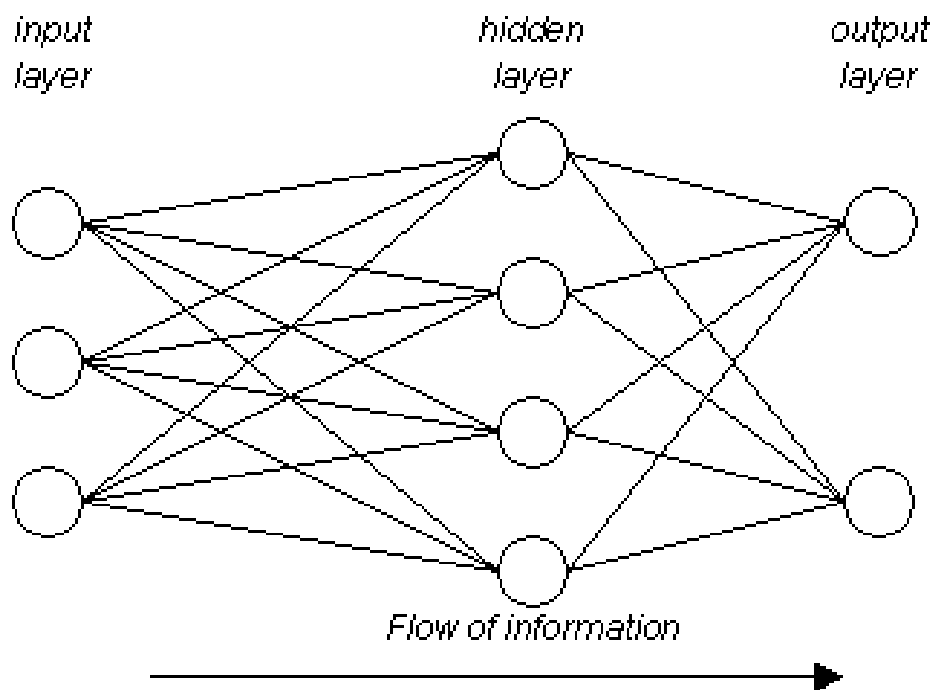
Within a region the site of greatest activation can be used to indicate how strongly similar the presented data is to the data the region has been trained to detect. Activation at or near the centre would be more certain than peripheral activation. This can be used to detect blurred or partially obscured characters, or to detect how similar a previously unseen character is to those shown in the training set.

This method has been successfully used to read the licence plate numbers of cars on petrol forecourts from surveillance camera recordings, which are notoriously noisy sources of data.

Multi-layer Networks

Another common type of network is a multiple-layer network. These are made up of at least three layers. The input layer, which is the raw input data, connected to a hidden layer of neurons, which is further connected to an output layer of neurons. The number of hidden layers can be increased, and is one of

the parameters that can be adjusted when trying to train a network



More complicated algorithms are employed to train these networks, the most common of which is called the back-propagation algorithm. This works in a similar way to the algorithm for training a single neuron, but it propagates the weight changes back into the hidden layers. Networks like this can solve problems that are not linearly separable, and are very good at calculations that need patterns to be detected in noisy data, for example detecting the metabolic profile of a particular disease from the metabolites in urine samples.

Uses of Artificial Neural Networks

Neural networks can, in principle, compute any computable function. In particular neural networks are suited to problems where the input data is unavoidably going to be noisy or where there is a lot of available data that cannot be easily solved using mathematical techniques.

An example of such a situation is the evaluation of medical images for pathological diseases. A recent breast cancer study used a neural network trained on ultrasound images of tumours that had been biopsied and diagnosed as malignant or benign. Physicians collected the images by identifying the location of the tumours on the ultrasound image, selecting that section and then storing it

for later use by the neural network. 140 training cases were stored, which were digitised and used to train the network, using the results of the biopsies to feedback the expected result. The fully trained network displayed a reasonably high level of accuracy and was helpful in guiding inexperienced operators to determine which patients should be given biopsies.

Bayesian Statistics

Tessella have written a separate supplement on this topic, which can be found at <http://www.tessella.com/Literature/Supplements/bayesianstats.htm>.

In brief, this technique is a statistical method that allows observed data to feed back into the process of statistical analysis. Where classical statistics takes sample data from a population and applies a statistical analysis method to the whole sample group, Bayesian statistical methods look at each case in turn and use the observed probabilities to update statistical assessment of the next cases. This allows the analysis to be performed in an iterative manner, which can help reduce the number of cases needed to produce statistical results.

If, for example, you wanted to analyse the dose-response curve of a drug, traditionally you would test the responses shown by a range of doses in large numbers of patients. Obviously the greater the number of patients who are involved in the trial, the more costly it is. Bayesian statistics can therefore offer an attractive alternative for assessing results. The initial drug dosages given can be analysed during the course of the trial and used to update the future doses given so that they are concentrated around the optimum dosage.

Hybrid Techniques

Each method described above can be used completely independently, or the techniques can be combined to exploit the strengths that each possess. There are numerous ways of hybridising these methods, of which three popular systems are explained below.

Neuro-fuzzy Systems

One powerful combination of soft computing techniques is the field of neuro-fuzzy systems. One of the main drawbacks of neural networks is that they are 'black boxes'. Data goes in, a calculation is performed and the answer is produced, but it is difficult to find out exactly what the calculation is doing or why. Fuzzy logic systems do not suffer from this problem, their rule sets can

always be ‘translated’ into easily understandable rules, but they are not capable of learning in the same way that neural networks are. This usually means that the rules and membership functions used need to be written and tuned by hand. When expert knowledge can be used to write these systems then this task is relatively simple, but this is not always possible to find experts in the problem domain.

Neuro-fuzzy systems address both these problems by applying the learning algorithms of multi-layer neural networks to adjust the distribution of the fuzzy logic sets used in a fuzzy logic system. This means that the fuzzy logical system can be created with no expert knowledge of the system that is being modelled, and by supplying the network with good training data, it will learn rules that can be used to model the system in general. These rules can be interpreted easily, making the system much more transparent and understandable.

This branch of Soft Computing has expanded rapidly in recent years, and it has been used in a wide variety of applications. In one such example a neuro-fuzzy system was trained to predict the next 48 hours of electricity demand in Victoria, Australia. When tested on previously unseen data, this system predicted the demand with an error rate of only 0.0092.

Genetic Fuzzy Systems

Another way of tuning fuzzy systems is to use Evolutionary Computing techniques to evolve the rule set and membership functions. This is particularly useful when a system cannot be trained using expected or previously measured output values, for example when a controller for a dynamic system is being developed. The controllers can, however, be judged on how well they perform, and this can be used as the fitness function in an evolutionary algorithm.

Much like the neuro-fuzzy system, the various aspects of the fuzzy logical system are parameterised. These aspects include the membership functions and information on the fuzzy rule set. This information is then encoded into a string or strings that represent a candidate’s genetic material. Genetic algorithms can then be applied to evolve the fuzzy system until an optimum solution is found.

The combination of these methodologies is a relatively recent development, but it has already been used in a number of applications. One success story is a system for diagnosing the cause of structural cracks in stone buildings. This is an area which requires a great deal of expert knowledge and which relies on multiple

observable features. The fuzzy logic system was used to take 42 of these observable features, and used fuzzy rules to determine the root cause from 6 possibilities. As the number of parameters in the system was so large, manually tuning the membership functions and the weights given to the rules to give the optimum results would have been difficult. For this reason a genetic algorithm was used, which encoded these values and evolved them, using the accuracy of the resulting system as the fitness function to guide the evolution. Once this tuning was complete the final system showed a good match with actual cases, showing an error rate of less than 4%

Using Neural Networks with Genetic Algorithms

One of the most important parts of a genetic algorithm is the specification of a good fitness function. If this is badly implemented then the evolved solutions will not be adapted to the task required. In many cases it is difficult to write a computationally efficient fitness function, as assessing the qualities of the individual designs can be an incredibly complex or non-linear problem. In cases like these, neural networks can be used to assess the fitness of solutions by training them on experimentally obtained data.

An example of this technique can be seen in the steel industry. Predicting the mechanical properties of steel alloys given their metal composition and tempering temperature is not very easy. Researchers at Sheffield University trained a neural network on experimental data to produce an effective predictor of alloy properties. This network was then used as the fitness function in a genetic algorithm to evolve new alloy designs with specific pre-subscribed properties. The genetic algorithm produced reliable and practical solutions to the design problem, which were verified later by a number of metallurgists.

When Should You Use Soft Computing?

As shown by the examples described above, soft computing can be used to address a very wide range of problems in all industries and business sectors. In general though, Soft Computing is a good option for complex systems, where:

- ❑ The system is non-linear, time-variant or ill defined.
- ❑ The variables are continuous
- ❑ A mathematical model is either too difficult to encode, does not exist or is too complicated and expensive to be evaluated

- ❑ There are noisy or numerous inputs
- ❑ An expert is available who can outline the rules-of-thumb that should determine the system behaviour.

General areas that need these kinds of systems are:

- ❑ Classification
- ❑ Optimisation
- ❑ Data mining
- ❑ Prediction
- ❑ Control
- ❑ Scheduling
- ❑ Decision support or auto-decision making

The most appropriate technique to use depends on the type of problem and the available data.

Neural Networks are good at classification, data-mining and prediction systems, where there is lots of available potentially noisy input data, which either needs to be classified into groups or which needs to be mapped to an expected output. They do, however, suffer from a lack of transparency, as the calculation is encoded in the weights and thresholds of multiply connected networks.

Evolutionary Computing techniques are adapted to optimisation and design problems, where a good solution can be recognised, but where there is no “correct” answer. This technique is essentially an efficient search technique that deals well with the problem of getting “stuck” with a sub-optimal solution. The main difficulty with the technique is defining how the solutions should be encoded and assessed for their fitness. Badly written fitness functions will result in answers that do not fit the problem being solved.

Fuzzy Logic systems are best suited to decision making and control systems that have “rules-of-thumb” that cannot be translated to hard mathematical formulae. These rules are used to perform a logical, non-linear mapping between inputs and outputs, which simulates the decision-making processes in humans. These systems rely very heavily on being able to determine rules that accurately

describe the system. They also need the membership functions for the fuzzy sets used to be tuned correctly, which can also be difficult due to their non-linear nature.

With such a large topic, it has only really been possible to scratch the surface of this technology and for further information the reader is referred to the websites listed below and to the numerous books now available on this area of computing.

References

1) A good introduction to evolutionary computing:

<http://www.cs.bham.ac.uk/Mirrors/ftp.de.uu.net/EC/clife/www/top.htm>

2) An introduction to fuzzy logic:

<ftp.cs.cmu.edu:/user/ai/pubs/faqs/fuzzy/fuzzy.faq>

3) An introduction to neural networks:

<ftp://ftp.sas.com/pub/neural/FAQ.html>

4) Tessella's technical supplement on Bayesian Statistics:

<http://www.tessella.com/Literature/Supplements/bayesianstats.htm>

5) J Felton, "*Survival of the Fittest in Drug Design*", Modern Drug Design, Nov/Dec 2000, Volume 3, No.9, pp. 49–50, 53-54.

6) Constantin von Altrock, "*Fuzzy Logic in Automotive Engineering*", Circuit Cellular INK, Issue 88 November 1997.

7) Dar-Ren Chen, MD, Ruey-Feng Chang, PhD and Yu-Len Huang, PhD, "*Computer-aided Diagnosis Applied to US of Solid Breast Nodules by Using Neural Networks*", Radiology. 1999;213:407-412.

Tessella Support Services plc
Creating Software for Science and Engineering

Tessella's services range from feasibility studies, through system design, development, implementation and ongoing support. Our expertise includes:

Data Analysis Software
Data Capture
Simulation Software
Advanced Graphics
Systems Support
Database Applications

Other Technical Supplements available include:

- | | |
|---|--|
| <input type="checkbox"/> Archiving of Electronic Info | <input type="checkbox"/> Object Oriented Programming |
| <input type="checkbox"/> Active Server Pages | <input type="checkbox"/> Pocket PC |
| <input type="checkbox"/> Automated GUI Testing | <input type="checkbox"/> Portable GUI Development |
| <input type="checkbox"/> Bayesian Statistics | <input type="checkbox"/> Printer Technology Guide |
| <input type="checkbox"/> Beowulf Clusters | <input type="checkbox"/> Real Time Systems |
| <input type="checkbox"/> C++ | <input type="checkbox"/> Regression Testing |
| <input type="checkbox"/> Client-Server Technology | <input type="checkbox"/> Security and the Internet |
| <input type="checkbox"/> COM | <input type="checkbox"/> Simulation |
| <input type="checkbox"/> Computational Fluid Dynamics | <input type="checkbox"/> Soft Computing |
| <input type="checkbox"/> Computer Image Processing | <input type="checkbox"/> Software Design Methodologies |
| <input type="checkbox"/> Decision Support Systems | <input type="checkbox"/> Software Development Cycle |
| <input type="checkbox"/> Electronic Data Capture | <input type="checkbox"/> Software Documentation |
| <input type="checkbox"/> Electronic Lab Notebooks | <input type="checkbox"/> Software Portability |
| <input type="checkbox"/> Excel | <input type="checkbox"/> Software Re-engineering |
| <input type="checkbox"/> Extending the Life of Software | <input type="checkbox"/> Software Specification |
| <input type="checkbox"/> Federal Drug Administration | <input type="checkbox"/> SQL |
| <input type="checkbox"/> FORTRAN 90 | <input type="checkbox"/> UNIX Inter-Process Comms |
| <input type="checkbox"/> Grid Computing | <input type="checkbox"/> UNIX Systems Performance |
| <input type="checkbox"/> High Throughput Screening | <input type="checkbox"/> UNIX Workstations |
| <input type="checkbox"/> Instrumentation | <input type="checkbox"/> Visual Basic 6 |
| <input type="checkbox"/> Integrated Lab Systems | <input type="checkbox"/> WAP |
| <input type="checkbox"/> J2EE | <input type="checkbox"/> Web Services |
| <input type="checkbox"/> Java | <input type="checkbox"/> Windows 2000 Services |
| <input type="checkbox"/> Lims | <input type="checkbox"/> XML |
| <input type="checkbox"/> Linux | <input type="checkbox"/> X Windows |
| <input type="checkbox"/> Microsoft Net | |



INVESTOR IN PEOPLE

Tessella Support Services plc

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

E-mail: info@tessella.com Web Address: <http://www.tessella.com>