Fuzzy Neural Network Interface: Development and Application

A thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering in Information Engineering at Massey University, Palmerston North, New Zealand.

Greg Todd

2003

Abstract

This project is concerned with the development and application of an interface for a fuzzy neural network (FuNN). The original program, for which the interface was written, is a tool to research the mapping of problem knowledge to initialize the weights of a FuNN. The interface concentrates on allowing the user to efficiently manipulate network settings and to be able to easily perform large numbers of experiments. After the interface was completed, the new integrated application was used to investigate the use of problem knowledge on FuNN training in specific image processing problems.

Acknowledgements

I would like to thank my supervisor Dr. Sanj Gunetileke for his tireless patience and help.

Contents

Chapter 1: Introduction

Overview

This thesis can effectively be broken down into two main sections:

The primary aim of this research is to produce a functional graphical user interface for a fuzzy neural network (FuNN) application, for use by academics and students studying the use of neural networks. The original program was developed by Gunetileke (2001), and though fully functional contained only a basic, mostly command line interface. The user interface will conform as closely as possible to the client requirements.

The secondary aim is to use the program to conduct experiments to determine the effect of using neural networks on two specific image-processing problems. The first problem will involve categorising various types of pollen. Textural and shape information, such as entropy, energy and area will be used to train the neural network, so it can identify each pollen type. The second problem will involve finding the best combination of inputs, into a neural network that will detect the iris edge from test images. Two or three significant inputs will be investigated to determine what effect they have on the final output.

Contents by chapter

Chapter 2 reports some of the relevant previous work in the field of FuNN's. This includes components of image processing, neural networks and Boolean logic.

Chapter 3 details the most important parts of the development of the interface for the SuperFuNN application program. It describes the life cycle development process, planning and analysis, requirements elicitation, design and design considerations.

In Chapter 4, the results of the interface development project are presented. This includes screen shots of the actual interface and descriptions of its features. It also includes information on user interface testing and function testing including an example unit and integration test.

Chapter 5 describes the first of the two image processing problems, iris edge detection. Although the iris edge detection problem has been looked at by previous researchers, this experiment looked at the effects of changing specific input variables to find the best combination for this problem.

Chapter 6 describes the problem of pollen classification using image processing and fuzzy neural networks. It describes a series of experiments dealing with the problem, and their results.

Chapter 7 looks at the usefulness of the interface, after having used it for real world problems. It also mentions ideas for future work to be done.

In Appendix 1, the list of pollen types with their scientific name, picture and there corresponding codes used during testing are displayed.

Appendix 2 contains Chapters One Two and Three of the user manual. Chapter one describes set-up and installation instruction, while Chapter Two is about 'Getting Started' with the SuperFuNN application. Chapter Three describes an actual example experiment. It also contains a list of common user mistakes and current application shortcomings.

In Appendix 3, the program code for the interface is listed. It includes all functions that were developed specifically for the interface, as well as those existing functions (from the original application) that have been significantly altered.

Chapter 2: Background

Introduction

This chapter will explain some of the background knowledge used during the course of this project. Fuzzy Neural networks (FuNN's) combine the fields of image processing, fuzzy logic and artificial neural networks.

Fuzzy Logic

Many of the terms people use in their description of objects and situations are difficult to model using mathematical equations. Often imprecise terms such as large, small, close, far, tall and short are used as descriptors in real world situations. However, these are difficult to translate into a mathematical form that might be suitable for input into computer programs. Such terms can be called 'fuzzy' because they cannot be sharply defined (Nguyen, 1997). Mathematical modelling of fuzzy concepts, now known as fuzzy logic, was presented by Zadeh in 1965. His contention was that meaning in natural language is a matter of degree. If we have the proposition that 'John is young', then it is not always possible to assert this is true or false. If John's age is x, then the compatibility of x with 'is young' is a matter of degree. This suggests that membership in a fuzzy subset should not be on a 0 or 1 (false or true) basis, but rather on a 0 to 1 scale. Figure 2.1 below shows how one might model the fuzzy concept of young, based on age (x), using the following model:

$$Y(x) = \begin{cases} 1 & \text{if } x < 40 \\ \dfrac{80 - x}{40} & \text{if } 40 \le x \le 60 \\ \dfrac{70 - x}{20} & \text{if } 60 < x \le 70 \\ 0 & \text{if } 70 < x \end{cases}$$

Figure 2.1 A fuzzy membership function for young.

Example of a Rule

Membership functions can be developed for both the antecedent (if ~) and consequent (then ~) parts of a rule. Figure 2.2 shows the membership functions for the antecedent and consequent of the following rule:

If a pollen is *big* then the likelihood it is a type 'A' is *high*.

Problem knowledge for current image processing problem can be described similarly.

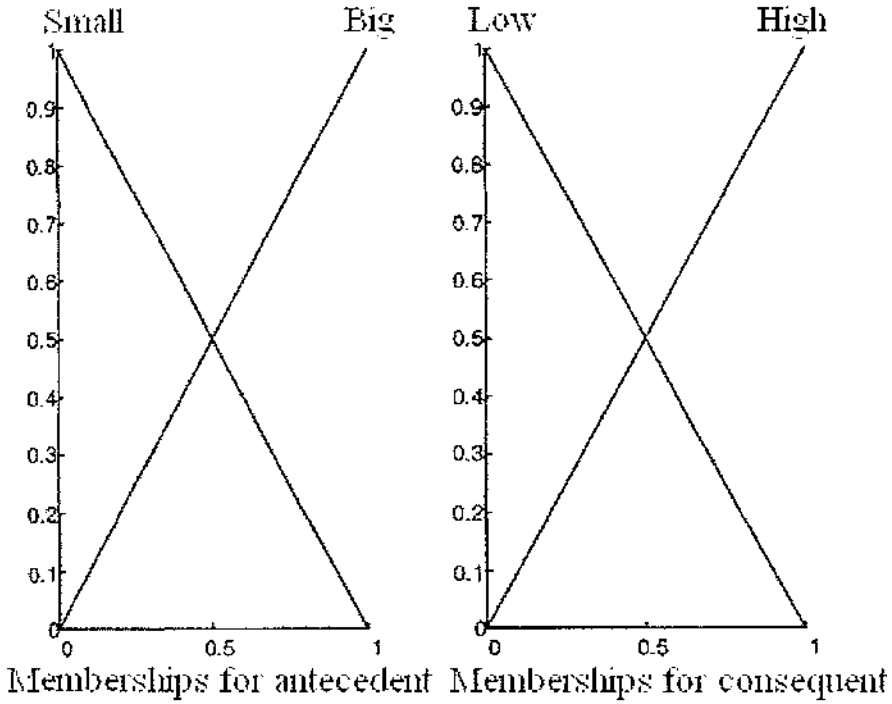

Figure 2.2 Membership function for an example rule

Artificial Neural Networks

The human brain is an example of a biological neural network. It consists of an enormously complex system of neurons, synapses, axons and dendrites. Haykin (1994) describes a neural network as a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. He notes that it resembles the brain in two respects:

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

Generally speaking, an artificial neural network tries to replicate the biological neural network through the use of computers. An artificial neural network, like the human brain, can learn by experience. A 'learning' neural network makes an iterative process of adjustments to adapt to its environment. There are three basic classes of learning paradigm: supervised learning, reinforcement learning and self-organised (unsupervised) learning. Supervised learning is performed under the supervision of an external teacher. Reinforcement learning is learning of input-output mapping through a process of trail and error. In unsupervised learning, there is no external teacher to instruct synaptic weight changes in the network (Haykin, 1994). The results change as a result of the nature of the data present. The FuNN developed by Kasabov (1996) used the supervised type training algorithm, as did Gunetileke (2001) when he developed his FuNN application program.

Usually the training weights of a neural network are initialised with small random numbers. Researchers have looked at using problem knowledge to help initialise a network. Proper initialisation is one of the prerequisites for fast convergence of a feed-forward network (Thimm *et al.*, 1997). There are various ideas that have been proposed for the initialisation of neural networks. Siroki (1998) and Gunetileke (2001) studied a problem knowledge based weight initialisation scheme. Rules representing problem knowledge are generated in an 'if ~ then ~' form, and used to

initialise the input weights into a network. The rules are implemented using a fuzzy neural network architecture.

Fuzzy Neural Networks

A Fuzzy Neural network is an architecture that combines neural networks and fuzzy logic (Kasabov, 1996). Kasabov notes that a FuNN consists of five layers, represented diagrammatically below in figure 2.2. For the input layer, a node represents an input variable as crisp values. These values are fed into the condition layer which performs the 'fuzzification' by triangular membership functions with the centres represented as the weights (Siroki, 1998). The values from the input layer that are fed into the condition layer are all in the range 0 – 1. Gunetileke (2001) notes that expert knowledge can be used to initialise the spacing of the membership functions. An important aspect of this layer is that different inputs can have different numbers of membership functions.

The output of the condition layer is passed to the rule layer, where each node represents a single 'fuzzy rule'. The rule layer is equivalent to the hidden layer of a multi layer perceptron (MLP) network. The activation of the rule layer node is the degree to which input data matches the antecedent component of an associated rule. Outputs from the rule layer are fed to the action element layer. In this layer each node represents a fuzzy label from the fuzzy quantification space of an output variable, for example "short", "medium" or "long". The activation of the node represents the degree to which this membership function is supported by the current data (Siroki, 1998).

Siroki (1998) explains that the output layer performs a modified centre of gravity defuzzification. Singletons representing centres of triangular membership functions are attached to the connections from the action to the output layer. Linear activation functions are used in this layer.

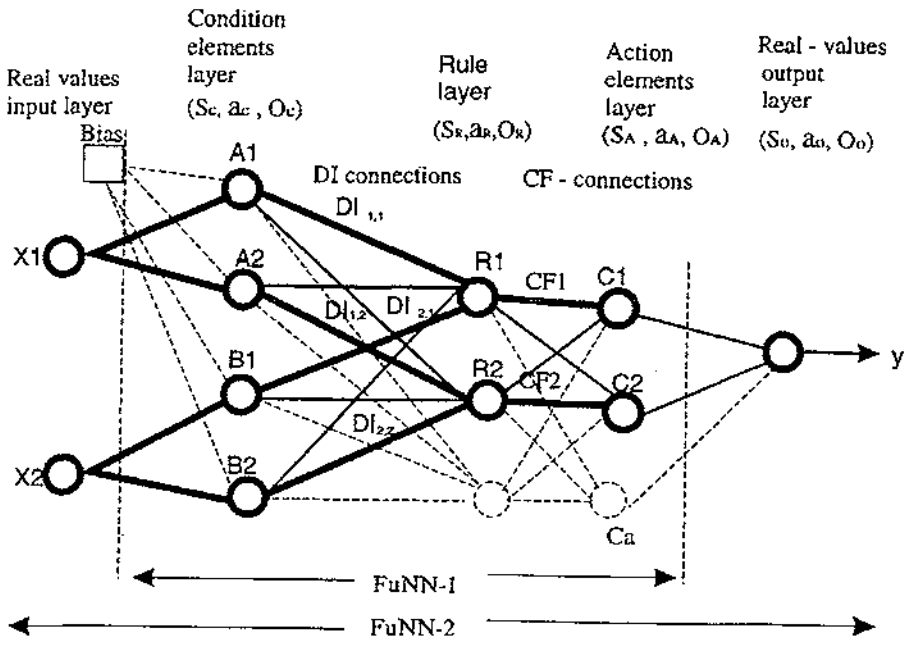

Figure 2.2 Architecture of a FuNN (Kasabov, 1996)

There are three methods to update the weights of a FuNN. The method used by Gunetileke (2001) for his FuNN application was, a partially adaptive version where the membership functions of the input and output variables do not change during training. The MLP section was trained using a modified back-propagation algorithm.

The training algorithm

This section describes the algorithm used for FuNN training as described in Gunetileke (2001). The algorithm is discussed in terms of neurons as they are the focus of the fuzzy architecture.

Forward Pass

The activation values for all the nodes in the network, from the first to the fifth layer are computated during this phase. A superscript indicates the layer and a subscript indicates a connection between layers.

Input Layer (first layer)

The input layer of neurons represents the input variables as crisp values. These values are fed into the condition layer which performs fuzzification.

Condition Layer (second layer)

This layer acts as a fuzzifier. The output from this layer is the degree to which the input belongs to the given membership function. The input weight to the condition node represents the centre for that particular membership function with the minimum and maximum determined using the centre of the adjacent membership functions. For the first and last membership functions for a variable, a shoulder is used instead. Each membership function is triangular and an input signal activates only two neighbouring membership functions simultaneously. The sum of the grades of these two membership functions for any given input is always one. For a triangular membership function the activation functions for a node i are:

If $a_i < x < a_{i+1}$ then $Act_i^c = 1 - \frac{x - a_i}{a_{i+1} - a_i}$

If $a_{i-1} < x < a_i$ then $Act_i^c = 1 - \frac{a_i - x}{a_i - a_{i-1}}$

If $x = a_i$ then $Act_i^c = 1$, where a_i is the centre of the triangular membership function.

Rule Layer (third layer)

The pre-condition matching of fuzzy rules is done through the connections from the condition layer to the rule layer. The connection weights in this layer may be set either randomly or according to a set of rules. The net input is given by the formula:

$$Net^r = \sum w_{rc} \times Act^c$$

while the activation is given by:

$Act^r = \frac{1}{1 + e^{(-g \times Net^r)}}$, where g is the gain factor.

Action Layer (forth layer)

The nodes in this layer and the connection weights function as those in the rule layer for net input and activation:

$$Net^a = \sum w_{ar} \times Act^r$$

$$Act^a = \frac{1}{1 + e^{(-g \times Net^a)}}$$

Output Layer (fifth layer)

Defuzzification to produce a crisp output value is performed in this layer. The centre of gravity (COG) defuzzification is used to convert from fuzzy to crisp.

$$Net^o = \sum w_{oa} \times Act^a$$

$$Act^o = \frac{Net^o}{\sum Act^a}$$

Backward Pass

The goal for this phase is to minimise the error function:

$$Error = \frac{1}{2} \sum (y^d - y^a)^2$$

where y^d is the desired output and y^a is the current output.

Hence the general learning rule (gradient descent) used is:

$$\Delta w \approx -\frac{\partial E}{\partial w}$$

$$w_{t+1} = w_t + \eta(-\frac{\partial E}{\partial w}) + \alpha(\Delta w_t)$$

where η is the learning rate and α is the momentum coefficient, and

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial Net} \times \frac{\partial Net}{\partial w} = -\delta \times Act$$

Thus, the weight update rule is:

$$\Delta w_{t+1} = \eta\delta \times Act + \alpha\Delta w_t$$

Output Layer (fifth layer)

When the weights are adapted the constraining rule is taken into account, which imposes restrictions to the change of the centres of the membership functions.

$$\delta^o = -\frac{\delta E}{\delta Net^o} = -\frac{\delta E}{\delta Act^o} \times \frac{\delta Act^o}{\delta Net^o} = y^d - y^a$$

Action Layer (forth layer)

The error for each node in this layer is calculated individually based on the output error and on the activation of this node having in mind the type of membership functions (triangular) used in the defuzzification layer as well as the type of defuzzification.

If $a_i < y < a_{i+1}$ then $d^a = 1 - \frac{y - a_i}{a_{i+1} - a_i}$

If $a_{i-1} < y < a_i$ then $d^a = 1 - \frac{a_i - y}{a_i - a_{i-1}}$

If $y = a_i$ then $d^a = 1$.

Hence,

$$\delta^a = f(Net^a) \times E^a = Act^a(1 - Act^a) \times (d^a - Act^a)$$

Rule Layer (third layer)

$$\delta^r = Act^r(1 - Act^r) \times \sum (w_{ar} \times \delta^a)$$

Condition Layer (second layer)

The weight w_{ic} is assigned as follows. If x^i lies in the fuzzy segment, then the corresponding weight should be increased directly proportionally to the propagated error from the previous layer as the error is caused by the weight. This proposition can be expressed by the following equation:

$$\delta^c = Act^c \times \sum (w_{rc} \times \delta^r)$$

Thus, the weight updating rule for this layer is:

$$w_{ic_{t+1}} = \eta \delta^c + \alpha \Delta w_{ic_t}$$

The new centres of the input triangular membership functions are also adjusted according to a partition range as for the output layer (Gunetileke, 2001).

Image Processing

The values fed into the FuNN can be gleaned from images through the implementation of image processing techniques. Image processing refers to the altering or analysing of images through a variety of techniques. When using computers, images are normally read in as matrix of numbers, each number (or set of three numbers for colour images) represents one pixel or 'dot' on the image. Mathematical operations can be performed on this 'matrix' to obtain information from it, and even change it.

Window filters can be used to obtain information about groups of pixels in the image as shown in figure 2.2 below. In this figure, the output image is defined as a function of the pixel in a window surrounding the equivalent position in the input image Gunetileke (2001). The filtering operation can be based on linear, non-linear or rule based functions.

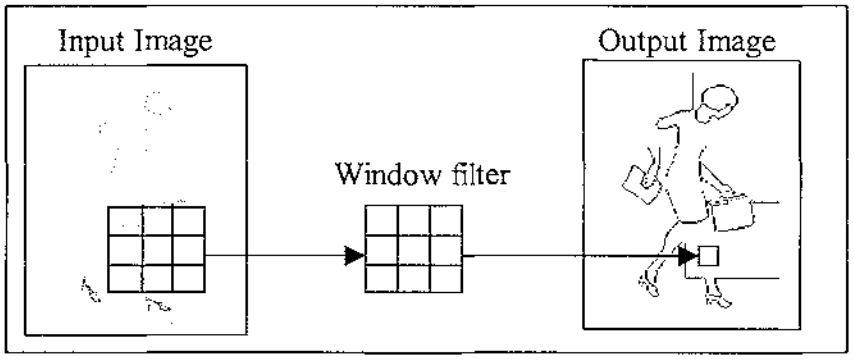

Figure 2.2 Operation of a window filter.

Often the selection of an appropriate window filter for image processing is an empirical process, which relies on skill and experience (Siroki, 1998). Sometimes it is possible to train a neural network to perform the filter operation. As most neural networks require both input and target variables to learn, so to does the neural network window filter (NNWF) (Gunetileke, 2001). Thus, for each image used for training the NNWF, an appropriate target image must be created.

Gunetileke (2001) describes the use of a fuzzy neural network window filter for use in image processing problems. The FuNNWF is represented visually in the figure 2.3 below.

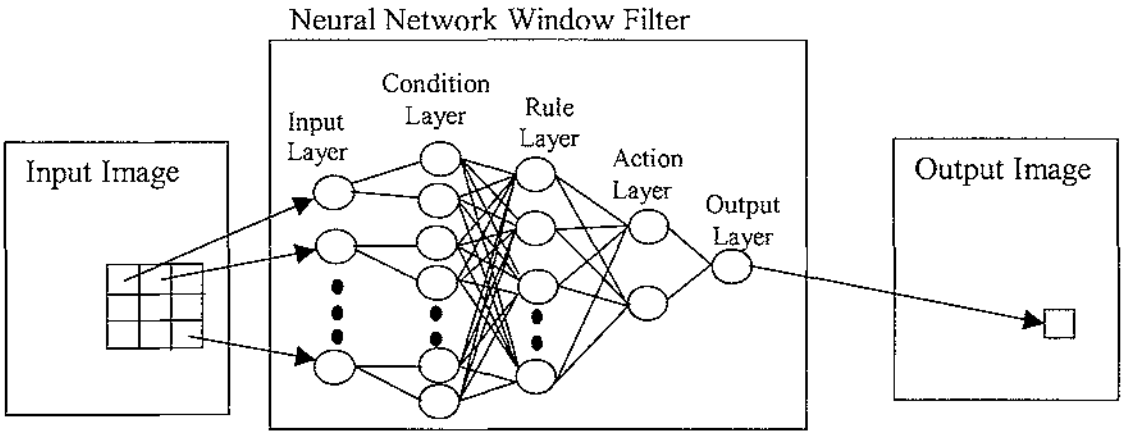

Figure 2.3 A Fuzzy neural network window filter.

Rule Mapping

Expert knowledge about a problem can be transformed into low level 'rules' that define the problem. These rules are then mapped to the weights of a FuNN during its initialisation to assist in its training. The main aims of the rules are to:

- Allow the network to start training from a position closer to the final solution
- To cover regions of the input-target space for which there is no training data
- To increase the robustness of the network

Two methods of mapping have been developed: Boolean logic rule mapping and conditional rule mapping. Boolean logic rule mapping effectively produces rules in the form:

If I1=M2 I2=M1 Then O1=M2

Where *I*- represents input values into the network, *M*- represents fuzzy membership values (for example M1=0, M2=1) and *O*- stands for network 'output'.

Conditional rule mapping produces rules of the form:

If I1<M2 I2>M3 Then O1=M2

Gunetileke (2001) notes that conditional rules are more powerful than Boolean logic rules. For a more detailed explanation of rule mapping see Gunetileke (2001) chapter 3.

Usually a 'complete' set of rules for describing a real world image processing problem cannot be derived. In this case the number of rule nodes in the network is made larger than the number of rules (creating 'free nodes'), so the network has enough freedom to adapt. The rules developed for real world image processing problems are unlikely to be 100 percent correct or complete. In this case the 'saturation value' associated with the rules can be adjusted according to the problem. A high saturation value represents high certainty in the correctness and completeness of each rule and effectively stops the training from changing the weights. A low saturation value means the rules are incomplete and/or uncertain and allows the training to alter the weight values relatively easily. Also, the 'quality factor' setting can be adjusted depending on the certainty of a rule. A high quality factor should be used for strong rules and a low quality factor used for weak rules. An incorrect rule with a high quality factor can have a degrading effect on network performance, as the correct rules and free nodes will try to compensate for them. As quality factor drops the weights associated with the incorrect rules can change more easily and the weights associated with the rule nodes change less (Gunetileke, 2001).