Complex Systems **6** (1992) 443–457

Training Artificial Neural Networks for Fuzzy Logic

Abhay Bulsari[*]
Kemisk-tekniska fakulteten, Åbo Akademi,
SF 20500 Turku/Åbo, Finland

Abstract. Problems requiring inferencing with Boolean logic have been implemented in perceptrons or feedforward networks, and some attempts have been made to implement fuzzy logic based inferencing in similar networks. In this paper, we present productive networks, which are artificial neural networks, meant for fuzzy logic based inferencing. The nodes in these networks collect an offset product of the inputs, further offset by a bias. A meaning can be assigned to each node in such a network, since the offsets must be either -1, 0, or 1.

Earlier, it was shown that fuzzy logic inferencing could be performed in productive networks by manually setting the offsets. This procedure, however, encountered criticism, since there is a feeling that neural networks should involve training. We describe an algorithm for training productive networks from a set of training instances. Unlike feedforward neural networks with sigmoidal neurons, these networks can be trained with a small number of training instances.

The three main logical operations that form the basis of inferencing—NOT, OR, and AND—can be implemented easily in productive networks. The networks derive their name from the way the offset product of inputs forms the activation of a node.

1. Introduction

Problems requiring inferencing with Boolean logic have been implemented in perceptrons or feedforward networks [1], and some attempts have been made to implement fuzzy logic based inferencing in similar networks [2]. However, feedforward neural networks with sigmoidal activation functions cannot accurately evaluate fuzzy logic expressions using the T-norm (see section 2.1). Therefore, a neural network architecture was proposed [3] in which the elementary fuzzy logic operations could be performed accurately. (For a good overview of fuzzy logic, see [4, 5].) A neural network architecture was desired for which the tedious task of training could be avoided, and

[*]On leave from Lappeenranta University of Technology. Electronic mail address: `abulsari@abo.fi`

in which each node carried a specific meaning. Productive networks, as then defined, offered many advantages (as described in [3]). It was shown that fuzzy logic inferencing could be performed in productive networks by manually setting the offsets. This procedure, however, encountered criticism, since there is a feeling that neural networks should involved training.

With minor modification—namely, the addition of the disconnecting offset—it is now possible to begin with a network of a size as large or larger than required, and train it with a few training instances. Because of the nature of productive networks, a small number of training instances suffices to train a network with many more parameters. The parameters must take the values -1, 0, or 1.

These modified productive networks retain most of the useful features of the previous design. It is still possible to manually set the offsets for well-defined problems of fuzzy logic. Each node still carries a meaning in the same manner as before. The networks are very similar to feedforward neural networks in structure. However, extra connections are permissible, which was not the case previously. A price has been paid for this increased flexibility, in the form of a more complicated calculation of the net input to a node.

2. The basic fuzzy logical operations

There is increasing interest in the use of fuzzy logic and fuzzy sets, for various applications. Fuzzy logic makes it possible to have shades of grey between the truth values of 0 (false) and 1 (true). Statements such as "the temperature is high" need not have crisp truth values, and this flexibility has permitted the development of a wide range of applications, from consumer products to the control of heavy machinery. Fuzzy expert systems are expert systems that use fuzzy logic based inferencing.

Almost all logical operations can be represented as combinations of NOT ($\sim$), OR ($\vee$), and AND ($\wedge$) operations. If the truth value of A is represented as $t(A)$, then we shall assume that

$$\begin{aligned} t(\sim A) &= 1 - t(A) \\ t(A \vee B) &= t(A) + t(B) - t(A)\,t(B) \\ t(A \wedge B) &= t(A)\,t(B) \end{aligned}$$

The OR equation shown above can be modified to a more suitable form as follows:

$$\begin{aligned} A \vee B &= \sim(\sim A \wedge \sim B) \\ t(A \vee B) &= 1 - (1 - t(A))(1 - t(B)) \end{aligned}$$

which is equivalent to the equation shown above, but is in a more useful form. Similarly, for three operands, one can write

$$\begin{aligned} t(A \wedge B \wedge C) &= t(A)t(B)t(C) \\ t(A \vee B \vee C) &= 1 - (1 - t(A))(1 - t(B))(1 - t(C)) \end{aligned}$$

Since Boolean logic is a special case of fuzzy logic (in which truth values are either 0 or 1), productive networks can be used for Boolean logic as well.

2.1 Unfitness of the sigmoid

We have yet to explain why fuzzy logic cannot be implemented in feedforward networks with sigmoidal activation functions. Such networks have a smooth transition from the "yes" to the "no" state, and are said to have a graceful degradation.

$A \wedge B$ can be performed in a feedforward neural network by $\sigma(t(A) + t(B) - 1.5)$, where σ is the sigmoid function from 0 to 1. This procedure has two major limitations. The truth value of $A \wedge B$ is a function of the sum of their individual truth values, which is far from the result of equations given above. This truth value is almost zero until their sum approaches 1.5, and after that it is almost one. The width of this transition can be adjusted, but the character of the function remains the same. If $t(A) = 1$ and $t(B) = 0$, $t(A \wedge B)$ is not exactly zero.

Another objection to the use of the sigmoid is more serious. Using $\sigma(t(A) + t(B) - 1.5)$ to calculate $t(A \wedge B)$ yields the following result:

$$t((A \wedge B) \wedge C) \neq t(A \wedge (B \wedge C))$$

3. Productive networks

A productive network, as defined here, no longer imposes a limit on the number of connections, as do feedforward networks. The extraneous connections do not matter since their weights can be set to zero. Each node plays a meaningful role in these networks. It collects an offset product of inputs, further offset by a bias. For example, the activation of the node shown in figure 1 can be written as

$$a = w_0 - |(w_1 - x_1)(w_2 - x_2)(w_3 - x_3)|$$

when the offsets are 0 or 1. If an offset is -1, it effectively disconnects the link. In general,

$$a = w_0 - \left| \prod_j (w_j - x_j)^{[1+\frac{1}{2}w_j(1-w_j)]} \right|$$

Thus, if an offset is -1, it only multiplies the product by 1. The output of the node is the absolute value of the activation, a.

$$y = |a|$$

The nondifferentiability of the activation function is not a problem. Nevertheless, if desired, the activation function can be made continuous by

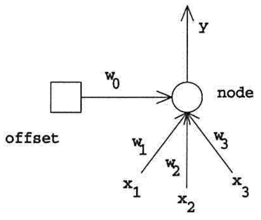

Figure 1: A node in a productive neural network.

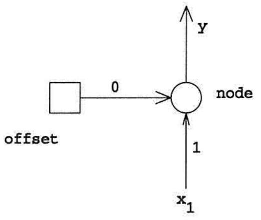

Figure 2: Inverting an input in a productive network

replacing it with a product of the argument and the -1 to 1 sigmoid, with a large gain, β:

$$y = a\left(-1 + 2\frac{1}{1+\exp(-\beta a)}\right)$$

The offset, w_j, shifts the value of the input by some amount, usually 0 or 1. The input remains unaffected when the offset is 0, and a logical inverse (negation) is taken when the offset is 1. In addition to the offset inputs, there is a bias, w_0, which further offsets the product of the offset inputs.

The productive network has several nodes with one or more inputs (see figures 6 and 8). The inputs should be positive numbers ≤ 1. Each of the nodes has a bias, alternatively called the offset of the node. A bias of 0 or -1 has the same effect—a node offset of zero is the same as not having a node offset. The output is a positive number ≤ 1. Productive networks are so named because of the multiplication of inputs at each node.

4. Implementation of the basic fuzzy logical operations

To show that one can represent any complicated fuzzy logic operation in productive networks, it suffices to show that the three basic operations can be implemented in this framework.

The simplest operation, NOT, requires an offset only, which can be provided by the input link (as shown in figure 2). Alternatively, this offset can be provided by the bias instead of the link, with the same result.

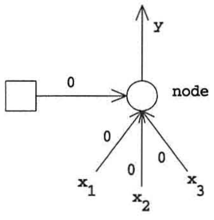

Figure 3: Applying AND to three inputs in a productive network

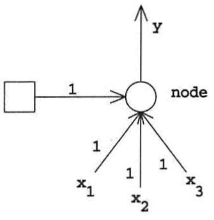

Figure 4: Applying OR to three inputs in a productive network

AND is also implemented in a facile manner in this framework. It needs only the product of the truth values of its arguments; hence, neither the links nor the bias have offsets (as shown in figure 3, for three inputs).

On the other hand, OR needs offsets on all the input links, as well as on the bias (see figure 4). The offsets for OR and AND indicate that they are two extremes of an operation, which would have offsets between 0 and 1. In other words, one can perform a 0.75 AND and a 0.25 OR of two operands A and B by setting $w_0 = 0.25$, $w_1 = 0.25$, and $w_2 = 0.25$.

Figure 5 shows how $\sim A \vee B$ can be implemented. This is equivalent to A implies B ($A \Rightarrow B$).

If the functions clarity(A) and fuzziness(A) are defined as

$$\begin{aligned}\text{clarity}(A) &= 1 - \text{fuzziness}(A)\\ \text{fuzziness}(A) &= 4 \times t(A \wedge \sim A),\end{aligned}$$

they can then be calculated in this framework.

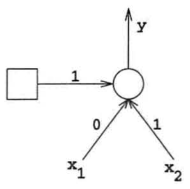

Figure 5: $\sim x_1 \vee x_2$

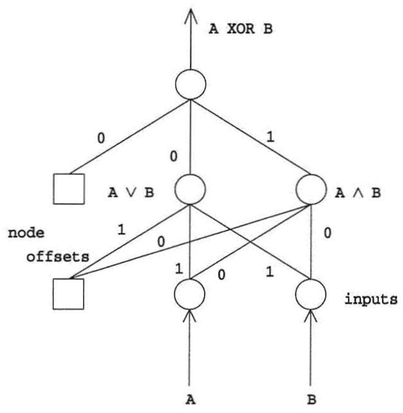

Figure 6: Network configuration for A XOR B

5. Illustrations of the manual setting of offsets

Applying AND and OR operations over several inputs can be performed by a single node. If some of the inputs must be inverted, this can be accomplished by changing the offset of the particular link. Thus, not only can a single node perform $A \wedge B \wedge C$ and $A \vee B \vee C$, but also $A \wedge B \wedge \sim C$ (which would require $w_0 = 0$, $w_1 = 0$, $w_2 = 0$, and $w_3 = 1$.) However, operations that require brackets for expression—for example, $(A \vee B) \wedge C$—require more than one node.

An exclusive OR applied two variables—A XOR B—can also be written as $(A \vee B) \wedge \sim(A \wedge B)$. Each of the bracketed expressions requires one node, with one more node required to perform AND between them (see figure 6). (A XOR B) can also be written as $(A \vee B) \wedge (\sim A \vee \sim B)$, or $(A \wedge \sim B) \vee (\sim A \wedge B)$, each of which could result in different configurations.

A program, SNE, was developed to evaluate the outputs of a productive network; an output for this XOR problem is shown in Appendix A. Feed-forward neural network studies often begin with this problem, fitting four points with nine weights by backpropagation. (Productive networks also require nine parameters, but fit the entire range of truth values between 0 and 1.) Figure 7 shows the XOR values for fuzzy arguments. $t(A)$ increases from left to right, $t(B)$ increases from top to bottom, and the values in the figure are ten times the rounded value of $t(A \text{ XOR } B)$.

In [1], a small set of rules was presented, designed to govern the selection of the type and mode of operation of a chemical reactor carrying out a single homogeneous reaction. As defined, there are regions in the state space (for example, between $r_0/r_1 = 1.25$ and 1.6) where none of the rules may apply. In fact, these were meant to be fuzzy regions, to be filled in a subsequent work such as this one. The set of rules has therefore been slightly modified, and is given below. There are two choices for the type of reactor, stirred-tank

```
00111122223333444455556666777788889999***
0011112222233334444555556666777778888999**
111112222333344444555566666777788888999*
11112222233333444455555666667777778888889999
1112222233333444445555566666777777888888999
112222233333444444555555666667777777888888899
2222233333334444445555556666677777777888889
222233333333444444555555666666777777777888888
222333333344444445555555666666677777777778888
2233333333444444455555555566666666777777777888
33333333444444455555555556666666666777777777778
333334444444455555555555566666666666777777777
333344444444455555555555666666666666677777777
3344444444455555555555556666666666666667777777
444444444455555555555555666666666666666666667
444444455555555555555666666666666666666666666
444445555555555555555566666666666666666666666
444555555555555555555666666666666666666666666
5555555555555555556666666666666666666666666
555555555555555556666666666666666666665555555
555555555555566666666666666666666655555555555
555555566666666666666666666666655555555555555
66666666666666666666666666665555555555555555
666666666666666666666666665555555555555555444
66666666666666666666666666555555555555544444
6666666666666666666666666555555555555544444444
76666666666666666666666655555555555554444444444
7777777666666666666666665555555555555444444444433
777777777766666666666666555555555544444444443333
777777777777666666666655555555554444444444433333
87777777777776666666666555555555544444444333333
8887777777777766666666665555555544444444333333322
888887777777777766666666555555544444443333333222
88888877777777766666666555555544444433333332222
98888888777777766666655555554444443333333322222
998888888777777766666555555544444433333322221 1
99988888888777776666655555544444433333222221111
999988888877777666665555544443333322222211111
*9999888888777776666655555444444333332222211111
**99998888877777666665555544444333332222211100
***9999888887777666665555444443333322222111100
```

Figure 7: *A* XOR *B*

or tubular; and two choices for the mode of operation, continuous or batch. In the following rules these choices are assumed to be mutually exclusive; that is, the sum of their truth values is 1.

1. If the reaction is highly exothermic or highly endothermic—say 15 kCal/gm mol (else we call it athermal)—select a stirred-tank reactor.
2. If the reaction mass is highly viscous (say 50 centipoise or more), select a stirred-tank reactor.
3. If the reactor type is tubular, the mode of operation is continuous.
4. If $r_0/r_1 < 1.6$, prefer a continuous stirred tank reactor.
5. If $r_0/r_1 > 1.6$, the reaction mass is not very viscous, and the reaction is quite athermal, prefer a tubular reactor.
6. If $r_0/r_1 > 1.6$, the reaction mass is not very viscous, but the reaction is not athermal, prefer a stirred-tank reactor operated in batch mode.
7. If $r_0/r_1 > 1.6$, the reaction mass is quite viscous, and the reaction is athermal, prefer a stirred-tank reactor operated in batch mode.
8. If the production rate is very high compared to the rate of reaction r_1 (say 12 m^3 or more), and $r_0/r_1 > 5$, prefer a stirred-tank reactor operated in batch mode.

9. If the production rate is very high compared to the rate of reaction r_1 (say 12 m^3 or more), and $r_0/r_1 < 5$, prefer a stirred-tank reactor operated in continuous mode.
10. If $r_0/r_1 \geq 1.6$, the reaction mass is quite viscous, and the reaction is not athermal, prefer a stirred-tank reactor operated in batch mode.

r_0 is the rate of reaction under inlet conditions, and r_1 is the rate of reaction under exit conditions. If their ratio is large, a plug-flow (tubular) reactor requires significantly less volume than a stirred-tank reactor operated continuously. A stirred-tank reactor operated in batch mode is similar to a plug-flow reactor when the length coordinate of the tubular reactor resembles time in a batch reactor. The aim of [1] was to investigate the feasibility of implementing a fuzzy selection expert system in a productive neural network; hence, the heuristics enumerated above are typical. They are not necessarily the best set of rules for selecting reactors for single homogeneous reactions; neither are they complete.

Figure 8 shows the implementation of these heuristics in a productive network. It is much simpler than a feedforward neural network; it requires no training, and does not have too many connections (weights). The offsets are either 0 or 1, unlike the weights (which can have any value between $-\infty$ and ∞). Of course, it also calculates the fuzzy truth values for the selection of type and mode of operation of a chemical reactor. It is a little more reliable since the function of each of the nodes in the network is known and understood, and there is no question of the sufficiency of the number of training instances. It may not be possible to represent every expression in two layers. Having several layers, however, is not problematic for productive networks, though it does cause difficulty in training feedforward neural networks.

It may be recalled that the inputs to productive networks are truth values between 0 and 1. The five inputs to the network shown in figure 8 are

A	$t(r_0/r_1 < 1.6)$
B	$t(\mu < 50)$
C	$t(\lvert\Delta H_{\mathrm{rxn}}\rvert < 15)$
D	$t(r_0/r_1 < 5)$
E	$t(F/r_1 < 12)$

These truth values can be calculated (using a ramp or a sigmoid) based on criteria for the width of fuzziness. (For example, for A, $t(r_0/r_1 < 1.2) = 1$ and $t(r_0/r_1 > 2.0) = 0$, with a linear interpolation in between.) Appendix B shows results of this system with clear inputs (truth values of 0 or 1). For confirmation, these results were fed into an inductive learning program. This program was able to elicit the heuristics for selecting a stirred-tank for continuous operation—$A \vee {\sim}B \vee {\sim}C \vee E)$ and $(A \vee (B \wedge C) \vee (D \wedge E)$. This, in effect, was the same as $(A \vee ({\sim}A \wedge B \wedge C) \vee (D \wedge E))$, implemented in the network directly from the heuristics. It was also confirmed that the implausible selection of a tubular reactor operated in batch mode never took place.

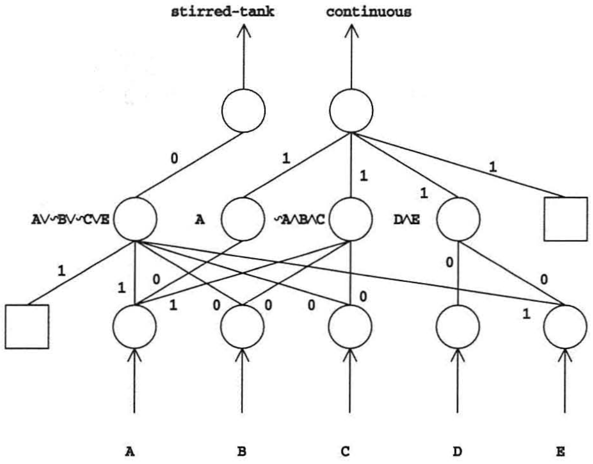

Figure 8: Fuzzy-selection expert system in a productive network

6. Limitations of productive networks lacking the disconnecting offset

If the inputs and outputs of a fuzzy logical operation are given, and if one wants a productive network to learn the correlation, it is almost impossible without the disconnecting offset. A productive network without the disconnecting offset (-1) cannot be easily trained. There is no way to switch off an output from a node that is connected. One must decide the connectivity beforehand—which can be, at best, good guesswork.

The productive network is intended primarily for representing fuzzy logical operations, and can of course do Boolean logic. That, however, is its limit. It has hardly any other application.

7. Illustrations of training productive networks

The training of neural networks is intended to reduce the sum of the squares of errors (SSQ) to a minimum, where the errors are the differences between the desired and the actual neural network outputs. In feedforward neural networks, it is sufficient to minimize the SSQ; in productive networks, however, it should be reduced to zero when the training instances are known to be accurate.

There are a finite number of possibilities for the weight vector. This is presumably an NP-complete problem. For a network with N weights, there are 3^N possibilities. It is therefore clear that searching the entire space is not feasible when N exceeds 7 or 8. To solve this problem we present a simple algorithm in the next subsection. Training is performed by modifying the offsets in a discrete manner (varying among -1, 0, and 1). The essence of this algorithm is similar to the Hooke and Jeeves method [6].

Since there is a disconnecting offset (-1), it is permissible to have more nodes than necessary. There is no over-parametrization in these networks; but an over-evaluation of fuzzy expressions is possible when there are extra nodes, and this can result in wrong outputs. The exact number of nodes can be determined if a complete set of training instances (all $2^{N_{\text{in}}}$ cases) is available; but this requires the approach of the previous work, which has been strongly criticized. Despite the fact that many respectable people in the field of neural networks think that training is necessary, or that "a network is a neural network only if it can be trained," we maintain that this is not the right way to construct productive networks.

7.1 The training algorithm

Hooke and Jeeves [6] proposed a zero-order optimization method, which looks for the direction in which the objective function improves by varying one parameter at a time. Derivatives are not required. When no improvement is possible, the step size is reduced. In the problem under consideration here, each parameter (offset) must take one of only three values. Hence, when a parameter is being considered for its effect on the objective function, the three cases are compared, and the one with the lowest SSQ is chosen.

Parameters to be considered for their effect on the SSQ can be chosen sequentially or randomly. With a sequential consideration of parameters, the search process often falls into cycles that are difficult to identify, stopping only at the limit of the maximum number of iterations. With a random consideration of parameters, the algorithm takes on a stochastic nature and is very slow, but does not fall into such cycles. Nevertheless, the sequential consideration is found to be preferable for the kinds of problems to be considered in sections 7.2 and 7.3.

Typical outputs of a program (SNT) implementing this algorithm are presented in Appendices C and D. The algorithm is fast and reliable. It runs into local minima at times (like any optimization algorithm would on a problem with several minima), but since it is relatively fast, it is easy to try different initial guesses. Because of the network architecture, it is possible to interpret intermediate results, even, presumably, at every iteration. The algorithm worked quite well with the problems considered in this paper.

7.2 The XOR problem

The XOR problem essentially consists of the training of a productive network to perform fuzzy XOR operation on two inputs. With eight training instances it was possible to train $(2, 2, 1)$ and $(2, 3, 1)$ networks easily, while $(2, 1)$ and $(2, 1, 1)$ networks could not learn the operation. Appendix C presents the results of training a $(2, 3, 1)$ network.

It is easy to interpret the results of training these networks. The $(2, 1)$ network learned $A \wedge B$ from the eight training instances. The SSQ was 1.038. The $(2, 1, 1)$ network learned $\sim A \wedge B$, resulting in the same SSQ, 1.038. These results are dependent on the initial guesses (to the extent that a different logical expression may result from a different initial guess), but $(2,1)$ and $(2,1,1)$ would not have an SSQ of less than 1.038. The $(2,2,1)$ network learned it perfectly, as $(A \vee B) \wedge ({\sim}A \vee {\sim}B)$. The $(2,3,1)$ network once ran into a local minimum, with SSQ $= 1.038$, expressing $({\sim}B) \wedge ({\sim}A \wedge {\sim}B) \wedge ({\sim}B)$. As shown in Appendix C, this network can also learn $(A \vee B) \wedge ({\sim}A \vee {\sim}B)$, ignoring one of the nodes in the hidden layer, which was ${\sim}(A \wedge B)$.

With four training instances (all clear inputs and outputs), the same networks were again trained by the algorithm described above. Unfortunately, there is no way to prevent the network from learning $(A \vee B)$ as $(A \wedge {\sim}B) \vee ({\sim}A \wedge B) \vee (A \wedge B) \vee B$. A is equivalent to as $A \vee A$ or $A \wedge A$ in Boolean logic, but not in fuzzy logic, given the way we have calculated conjunctions and disjunctions. Therefore, if one starts with clear training instances (no fuzzy inputs or outputs), the network may learn one of the expressions that is equivalent in Boolean logic. Fortunately, however, there was a tendency to leave hidden nodes unused. The $(2,2,1)$ network learned the XOR function as ${\sim}(A \vee {\sim}B) \vee (A \wedge {\sim}B)$. The $(2,5,1)$ network also learned it correctly as $({\sim}A \wedge B) \vee {\sim}({\sim}A \vee B)$ without adding extraneous features from the hidden layar (leaving 3 nodes unused).

7.3 The chemical reactor selection problem

The chemical reactor selection heuristics listed in section 5 can also be taught to productive networks from the 32 clear training instances (see Appendix B). The $(5,5,2)$, $(5,6,2)$, $(5,8,2)$, and $(5,10,2)$ networks were able to learn the correct expressions, with an SSQ of 0.00. The number of iterations required was between 200 and 3000, but each iteration took very little time. The typical run times on a microVax II were between 1 and 10 minutes.

The $(5,5,2)$ network used only four hidden nodes, the minimum required. The $(5,8,2)$ and $(5,10,2)$ networks used seven and eight nodes, respectively. They typically had one or two nodes simply duplicating the input (or its negation). The results obtained with the $(5,8,2)$ network are shown in Appendix D.

7.4 Limitations of the training approach

In Boolean logic, $(A \vee B)$ can also be written as $(A \wedge {\sim}B) \vee ({\sim}A \wedge B) \vee (A \wedge B) \vee B$. A is equivalent to as $A \vee A$ or $A \wedge A$ in Boolean logic, but not in fuzzy logic, given the way we have calculated conjunctions and disjunctions. Therefore, if one starts with clear training instances (no fuzzy inputs or outputs), the network may learn one of the expressions that is equivalent in Boolean logic.

It is not easy to obtain training instances with fuzzy outputs. It is always possible to set the offsets manually, once the logical expression is known. Training is apparently an NP-complete problem. Networks cannot be trained sequentially like, as in backpropagation.

8. Conclusions

Productive networks, intended for fuzzy logic based inferencing, have fuzzy truth values as inputs. The nodes collect an offset product of the inputs, further offset by a bias. A meaning can be assigned to each node, and thus one can determine the number of nodes required to perform a particular task. Training is not required for problems of fuzzy logic, but the offsets are fixed using the problem statement.

The three primary logical operations (NOT, OR, and AND), which form the basis of inferencing, can be implemented easily in productive networks. The offsets are either 0 or 1, and the nodes do not have complete connectivity across adjacent layers (as do feedforward neural networks).

Productive networks are much simpler when compared to feedforward neural networks. They require no training, and do not have too many connections (weights). Their offsets are either 0 or 1 (unlike the weights, which can have any value between $-\infty$ and ∞). They are a little more reliable, since the function of each of the nodes in the network is known and understood. Having several layers is not problematic for productive networks. A disconnecting offset of -1 permits training, which can be accomplished with very few instances. One need not know the number of nodes required.

Due to the significant advantages of productive networks over feedforward networks, they could find more widespread use in problems involving fuzzy logic.

Appendix A. A typical output from SNE

```
Number of input and output nodes :           2     1
Number of hidden layers :    1
Number of nodes in each hidden layer :     2
Number of offsets :      9
Print option 0/1/2 :    1
Offsets taken from file :    xor.in

 0.5000  0.5000  0.5625

layer 2  0.5625
layer 1  0.7500  0.2500
layer 0  0.5000  0.5000

 0.7500  0.2500  0.6602

layer 2  0.6602
layer 1  0.8125  0.1875
layer 0  0.7500  0.2500

 0.2500  0.7500  0.6602
```

```
layer 2  0.6602
layer 1  0.8125  0.1875
layer 0  0.2500  0.7500

 0.3333  0.3333  0.4938

layer 2  0.4938
layer 1  0.5556  0.1111
layer 0  0.3333  0.3333
```

Appendix B. Results of reactor selection with clear inputs

```
Number of input and output nodes :              5     2
Number of hidden layers :    1
Number of nodes in each hidden layer :      4
Number of offsets :    20
Print option 0/1/2 :    0
Offsets taken from file :    SEL.IN

 0    0    0    0    0    1    0
 0    0    0    0    1    1    0
 0    0    0    1    0    1    0
 0    0    0    1    1    1    1
 0    0    1    0    0    1    0
 0    0    1    0    1    1    0
 0    0    1    1    0    1    0
 0    0    1    1    1    1    1
 0    1    0    0    0    1    0
 0    1    0    0    1    1    0
 0    1    0    1    0    1    0
 0    1    0    1    1    1    1
 0    1    1    0    0    0    1
 0    1    1    0    1    1    1
 0    1    1    1    0    0    1
 0    1    1    1    1    1    1
 1    0    0    0    0    1    1
 1    0    0    0    1    1    1
 1    0    0    1    0    1    1
 1    0    0    1    1    1    1
 1    0    1    0    0    1    1
 1    0    1    0    1    1    1
 1    0    1    1    0    1    1
 1    0    1    1    1    1    1
 1    1    0    0    0    1    1
 1    1    0    0    1    1    1
 1    1    0    1    0    1    1
 1    1    0    1    1    1    1
 1    1    1    0    0    1    1
 1    1    1    0    1    1    1
 1    1    1    1    0    1    1
 1    1    1    1    1    1    1
```

Appendix C. A typical output from SNT

```
This is the output from SNT.

Number of input and output nodes :               2     1
Number of hidden layers :    1
Number of nodes in each hidden layer :      3
Number of offsets :    13
Print option 0/1/2 :    1
The offsets taken from WTS.IN :
 -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
Number of iterations :      500

Number of patterns in the input file     8
The offsets W >>> 1 to  13 :
  1   0   0   1   1   1   0   0   0   0  -1   0   1

The residuals F >>> 1 to    8 :
   0.00000E+00
   0.00000E+00
   0.00000E+00
   0.00000E+00
   0.00000E+00
   0.43750E-04
   0.43750E-04
   0.00000E+00

SSQ :  0.38281E-08
```

Appendix D. An output from SNT for the chemical reactor selection heuristics

```
This is the output from SNT.

Number of input and output nodes :               5     2
Number of hidden layers :    1
Number of nodes in each hidden layer :      8
Number of offsets :    66
Print option 0/1/2 :    0
Seed for random number generation :    10201
```

```
Number of iterations :    1000

Number of patterns in the input file    32
The offsets W >>> 1 to  66 :
  1   1   0   0  -1  -1   0   1   0  -1  -1   0   1   0   0  -1
  1   1   0  -1  -1  -1   0   0  -1   0   0   1  -1   1  -1   1
  1  -1  -1   0   0   1  -1  -1  -1  -1  -1   0   1  -1  -1   0
  1   1   1   0   1  -1  -1   0  -1   1   0  -1   0   1   0   0
  0   0

SSQ :  0.00000E+00
```

References

[1] A. B. Bulsari and H. Saxén, "Implementation of Chemical Reactor Selection Expert System in a Feedforward Neural Network," *Proceedings of the Australian Conference on Neural Networks*, Sydney, Australia, (1991) 227–229.

[2] S.-C. Chan and F.-H. Nah, "Fuzzy Neural Logic Network and Its Learning Algorithms," *Proceedings of the 24th Annual Hawaii International Conference on System Sciences: Neural Networks and Related Emerging Technologies*, Kailua-Kona, Hawaii, **1** (1991) 476–485.

[3] A. Bulsari and H. Saxén, "Fuzzy Logic Inferencing Using a Specially Designed Neural Network Architecture," *Proceedings of the International Symposium on Artificial Intelligence Applications and Neural Networks*, Zurich, Switzerland, (1991) 57–60.

[4] L. A. Zadeh, "A Theory of Approximate Reasoning," pages 367–407 in *Fuzzy Sets and Applications*, edited by R. R. Yager et al. (New York, Wiley , 1987).

[5] L. A. Zadeh, "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems," *Fuzzy Sets and Systems*, **11** (1983) 199–227.

[6] R. Fletcher, *Practical Methods of Optimization. Volume 1, Unconstrained Optimization* (Chichester, Wiley, 1980).