This paper describes a method for growing a recurrent neural network of fuzzy threshold units for the classification of feature vectors. Fuzzy networks seem natural for performing classification, since classification is concerned with set membership and objects generally belonging to sets of various degrees. A fuzzy unit in the architecture proposed here determines the degree to which the input vector lies in the fuzzy set associated with the fuzzy unit. This is in contrast to perceptrons that determine the correlation between input vector and a weighting vector. The resulting membership value, in the case of the fuzzy unit, is compared with a threshold, which is interpreted as a membership value. Training of a fuzzy unit is based on an algorithm for linear inequalities similar to Ho-Kashyap recording. These fuzzy threshold units are fully connected in a recurrent network. The network grows as it is trained. The advantages of the network and its training method are: (1) Allowing the network to grow to the required size which is generally much smaller than the size of the network which would be obtained otherwise, implying better generalization, smaller storage requirements and fewer calculations during classification; (2) The training time is extremely short; (3) Recurrent networks such as this one are generally readily implemented in hardware; (4) Classification accuracy obtained on several standard data sets is better than that obtained by the majority of other standard methods; and (5) The use of fuzzy logic is very intuitive since class membership is generally fuzzy.

1. Introduction

1.1. Background

The advantages of the network and its training method, described here, are due to the recurrent nature of the network, the fuzzy property of the threshold units, and the fact that the training algorithm allows the network to grow to the required size. Recurrent networks are more readily implemented in hardware. Growth of a network prevents development of a network that is otherwise poor in generalisation, because it remembers its training data. Fuzzy logic is very intuitive because set membership in reality is generally fuzzy.

A class of neural networks which has proven to be useful for classification is the class of recurrent neural networks, an example of which is the Hopfield network. Recurrent networks are inherently more effective than feedforward networks because they are able to dynamically store and use state information indefinitely due to built-in feedback. They can also be modelled and trained to function like deterministic, sequential finite state automata.

A Hopfield network is fully recurrent except that no self-connections are allowed and other connections are symmetric. Several researchers have used the Hopfield network or version thereof, as a content addressable memory for the purpose of pattern classification. One of the main advantages of the Hopfield network versus MLP, for classification, is

*E-mail: rbrouwer@cariboo.bc.ca
the simplicity of the learning procedure compared to back propagation. Networks of this type have beneficial characteristics such as: (a) simple structure; (b) strong self-organising features; and (c) rapid learning rate. They are readily implemented in inexpensive hardware in extremely large sizes because of their simple structure. This type of network may be used to store memories and can therefore be used to store the exemplars or training elements corresponding to classes. They have found many applications in image processing, optimization and pattern recognition. A feature of the Hopfield network that is desirable for pattern recognition is that these networks can recover desired features from distorted patterns.

Since class boundaries are generally fuzzy, it is intuitive to incorporate fuzziness in classification methods. Classification is concerned with set membership and objects that generally belong to sets of various degrees; therefore a fuzzy neural network seems natural for doing classification. Numerous papers written on fuzzy neural networks demonstrate the fact that it is useful to combine fuzzy logic with neural networks. Algorithms can be fuzzified, perceptrons can be fuzzified or networks can be used to carry out the mapping of fuzzy rules.

Another aspect of neural network techniques is that of network growing techniques and network pruning techniques. Network growing techniques consist of starting with a small number of hidden units, and then adding new units, one by one, to gradually refine the network. There is a class of learning algorithms that grow a multilayer feedforward network to classify any linearly nonseparable clusters of patterns. These algorithms are used to learn the connection weights and the topology of the network simultaneously. For feedforward networks, growing techniques consist of gradually expanding the hidden layer. In case of feedforward networks with hidden layers, and a recurrent network is like a feedforward network with the same connection matrix for each layer, it is desirable to control the number of hidden nodes so that the optimal approximation/generalisation accuracy is delivered. The criteria for growing are such that it best enhances the accuracy. Simpler architectures with fewer parameters have a better chance of avoiding the problem of over-fitting or over-learning, which is frequently encountered in complicated adaptive systems. In addition to improving generalisation, a smaller number of free parameters reduces storage requirements and execution time for classification.

1.2. Brief overview of architecture of network

This paper describes the growth of a fully recurrent Hopfield style artificial neural network with fuzzy threshold units (FTU) referred to as a Fuzzy Recurrent Artificial Neural Network (FRANN). The learning algorithm generates values for a connection matrix, which grows during training, that allows given patterns to be stored in the network as fixed points or stable states and allows these fixed states to be retrieved on the basis of noisy versions of these fixed states. A training algorithm based on an algorithm for linear inequalities described by Ho and Kashyap is used to determine the fuzzy sets for each unit. The network grows, during training, to the size required to do the classification.

A fuzzy threshold unit, described later, determines the degree to which the input vector lies in the fuzzy set associated with the fuzzy unit. The results of the comparisons are then combined using a generalised mean function to produce a single number. This may be compared to determine the correlation between an input vector and the weighting vector associated with the unit as in the case of perceptrons. The resulting membership value in the case of the fuzzy unit is compared to a threshold, which is interpreted also as a membership value. The behaviour of a typical binary fuzzy unit whose input is restricted to binary vectors is based on fuzzy logic in that each component of the binary input vector to the fuzzy unit is compared to a number which represents the membership value for a 0 in that position. The results of the comparisons for a typical fuzzy unit are then combined using an aggregation function to produce a single number, which is compared to a threshold. Training of a fuzzy unit is based on an algorithm for linear inequalities similar to Ho-Kashyap recording.

These FTU’s are fully connected in a recurrent network. The connection matrix is initialised so that all membership values are equal to 0.5, which is equivalent to having just one unit since there is essentially only one unique membership value. As the network is trained, the number of unique
membership values increases which is equivalent to increasing the number of distinct units. Thus the network grows as it is trained. Without growing techniques, the connection matrix that results from training the network generally has a great deal of redundancy in that there are few unique rows and columns compared to the dimensions of the matrix. The membership matrix for classification of feature vectors using binary representations could easily require more than 10,000 elements unless the number of unique connections was kept to a minimum. Thus, reductions in computational intensity and storage requirements through growing techniques are welcomed.

FRANN is similar to a Hopfield network in that it is fully recurrent. It differs from the Hopfield network in that symmetry is not required, self connections are allowed, changes in state are synchronous where all units change state at the same time and most significant of all, the connections between units which make up the input lines to a unit define 2-element fuzzy sets rather than weights.

The paper commences with a review of fuzzy sets and then goes on to describe a fuzzy unit with its training algorithm. Next is a description of a recurrent network of FTU’s and its training algorithm for storing mappings and thereby doing classifications. This is followed by a description of utilising the sparseness in the connection matrix during retrieval of attractors. The paper concludes with the results of doing the simulations.

2. Fuzzy Sets

Fuzzy sets are means for expressing the vagueness of inexact information. These sets can be thought of as a generalisation of classical set theory and were introduced by Zadeh as a mathematical way to represent vagueness in linguistics. Elements may belong to a fuzzy set to various degrees. For example, the set of tall men is a fuzzy set. A man of height 6’ belongs to this fuzzy set to a lesser degree than one of height 6’3”. The opposite of a fuzzy set is a crisp set. The characteristic function for a crisp set $C$ with a universe of discourse $X$, described by $C : X \rightarrow \{0, 1\}$, is replaced by a membership function $\mu_C : X \rightarrow [0, 1]$ for a fuzzy set $F$ and the same universe of discourse $X$. $\mu_F(x)$ denotes the membership value of $x$ in $F$ and expresses the degree to which $x$ is compatible with the concept expressed by $F$. The $\alpha$-cut of fuzzy set $F$ is the set of all members of $F$ whose membership values are greater than or equal to $\alpha$. When $X$ is finite, fuzziness of set $A$, $f(A)$, is measured by the sum over all elements of $X$ by

$$\sum 1 - |2\mu_F(x) - 1|.$$ (1)

Note that maximum fuzziness for a single element is 0.5. Maximum fuzziness for a set is obtained if all elements have a membership value of 0.5 in which case it is equal to the number of elements in the set. Zero fuzziness is obtained if all elements have a membership value of 0 or 1.

3. Fuzzification of a Perceptron

A perceptron is an artificial neuron with a particular transformation function. It is described by the transform

$$\text{sgn}(w \cdot x - t),$$ (2)

where $x \in \{-1, 1\}^n$ is the input vector, $w \in \mathbb{R}^n$ is the weight vector, $t \in \mathbb{R}$ is the threshold and $\text{sgn}(s) = (s \geq 0) - (s < 0)$. The latter is called the threshold function. Bolded numbers and letters are generally vectors in this document. The convention that is used here is that false is 0 and true is 1. The value of the weight vector may be obtained by the perceptron learning algorithm, also called Rosenblatt learning algorithm or by the delta rule.

For the fuzzy threshold units in FRANN, the weights, $w_i$, rather than being fuzzified are replaced by membership functions $\mu_i$. In the transformation performed by the fuzzy unit, a single weighted input, $x_i$ times $w_i$ is replaced by a membership value $\mu_i(x_i)$. The individual membership values are combined to produce a single value in $[0, 1]$ which may be interpreted as a single membership value for the entire vector $x$. The fuzzy unit, as shown in Fig. 1, determines the degree to which the input vector belongs to the fuzzy set defined by the unit. The resulting value is then compared to a threshold value $t$.

The fuzzy unit decides which input vectors belong to the $t$-cut of the fuzzy set defined by the unit and thereby does a two-way classification as perceptrons do. The fuzzy values of the individual inputs are aggregated by the generalised mean, which for a
The component values of $v$ are in the range $[0, 1]$ and therefore the expression above will have a value in $[0, 1]$. $L$ may be interpreted as the combined membership value of the input vector in some fuzzy set defined by the fuzzy unit. In this paper, a value of $r = 2$ is used so that expression 3 becomes $\frac{\|v\|}{\sqrt{n}}$. The output of the unit is defined by the binary valued expression $(L \geq t)$. A threshold, $t$, for the quantified output, would have to be an element of $(0, 1]$, since the membership value for an input vector must be greater than or equal to $t$ to produce a 1 at the output and a threshold of 0 would mean all input produces a 1 for output. The threshold defines a $t$-cut for the fuzzy set consisting of all the possible input vectors to the fuzzy unit.

For connecting the fuzzy units to produce a fully recurrent network, the output of each unit must form part of the inputs to other units and itself. Therefore the inputs must be binary if the outputs are to be binary. With this in mind, the fuzzy unit, as shown in Fig. 2, is further restricted so that the domain is $\{0, 1\}^n$.

The universe of discourse, $S_i$, corresponding to an input line of a fuzzy unit, now has only two members. Let the membership value for the value 0 be denoted by $m_i$ and let the membership value for 1 on the $i$th input line be the complement with respect to 1, i.e. $1 - m_i$. The effect of this is that both values are between 0 and 1, and that the values are negatively correlated. Maximum fuzziness occurs when the fuzziness for both 0 and 1 is 0.5. Each fuzzy set membership function is now completely defined by one value $m_i$, just as in the case of weights. Since $\mu_i(0) = m_i$ and $\mu_i(1) = 1 - m_i$, the membership function can be expressed as:

$$\mu_i(x_i) = |m_i - x_i|.$$

If the generalised mean with $r = 2$ is used for aggregation, the output of the fuzzy unit before thresholding becomes:

$$L = \frac{\|m - x\|}{\sqrt{n}}. \quad (5)$$

Assuming that $|m - x|$ is the vector of membership values corresponding to input vector $x$, and $L$ is the
mean length of the membership vector. Note that \( \mathbf{m} \) is the membership vector for input \((0, 0, \ldots, 0)\).

4. Training Algorithm for an FTU

The training algorithm for obtaining the membership values that produce the desired outputs for given inputs may be described as follows. Let \( G(\mathbf{m}, \mathbf{x}) \equiv L(\mathbf{m}, \mathbf{x}) - t \), let \( d(\mathbf{x}) \) be the desired output for input vector \( \mathbf{x} \) and let \( T(\mathbf{m}, \mathbf{x}) \equiv G(\mathbf{m}, \mathbf{x}) \geq 0 \). The objective is that \( T(\mathbf{m}, \mathbf{x}) = d(\mathbf{x}) \) for all input patterns \( \mathbf{x} \), i.e. that \( T(\mathbf{m}, \mathbf{x}) \) and \( d(\mathbf{x}) \) have the same truth value denoted by 0 for false and 1 for true. This is equivalent to:

\[
G(\mathbf{m}, \mathbf{x}) \geq 0 \quad \text{for} \quad d(\mathbf{x}) = 1
\]

and

\[
G(\mathbf{m}, \mathbf{x}) < 0 \quad \text{for} \quad d(\mathbf{x}) = 0.
\]

This is satisfied if the following is true. \(^{45} \)\( t \) will be considered as a constant in the following:

\[
J = G(\mathbf{m}, \mathbf{x})(2d(\mathbf{x})-1)-b = 0 \quad \text{where} \quad b > 0. \quad (7)
\]

Because \( J^2 \) is always nonnegative, hence reducing it will make it approach 0 and it is useful to find its gradients with respect to \( \mathbf{m} \) and \( b \).

\[
\nabla J^2 = 2J
\n\n\nabla J
\]

Since the gradient of \(|\mathbf{m} - \mathbf{x}|\) with respect to \( \mathbf{m} \) is the unit vector in the direction of \( \mathbf{m} - \mathbf{x} \), i.e. \((\mathbf{m} - \mathbf{x})/||\mathbf{m} - \mathbf{x}||\).

\[
\nabla \mathbf{m} J = ((2d(\mathbf{x}) - 1)/\sqrt{n})(\mathbf{m} - \mathbf{x})/||\mathbf{m} - \mathbf{x}||. \quad (9)
\]

Also

\[
\nabla b J = -1,
\]

where \( \mathbf{m} \) and \( b \) can be found by modifying them using the equations:

\[
\Delta \mathbf{m} = -2c_1 J \nabla \mathbf{m} J
\]

\[
= -2c_1/\sqrt{n} J(2d(\mathbf{x}) - 1)(\mathbf{m} - \mathbf{x})/||\mathbf{m} - \mathbf{x}||; \quad (11)
\]

\[
\Delta b = -2c_2 J \nabla b J = 2c_2 J , \quad (12)
\]

\[
\text{while} \quad \text{desired accuracy has not yet been achieved, do the following for one or more epochs;}
\]

\[
\text{for} \quad k \in \text{random permutation of } 0, 1, \ldots, p-1
\]

\[
\text{Modify}_m \text{and}_b(m, b(k), x^{(k)}, m, b(k))
\]

\[
\text{endfor}
\]

\[
\text{endwhile}
\]

Fig. 3. Training algorithm for an FTU.

Procedure Modify\(_m\)\_and\(_b\) (in:\(m, b, x; \)out \(m, b\))

\[
G(\mathbf{m}, \mathbf{x}) \leftrightarrow ||\mathbf{m} - \mathbf{x}||/\sqrt{n} - t
\]

\[
T(\mathbf{m}, \mathbf{x}) \leftrightarrow G(\mathbf{m}, \mathbf{x}) \geq 0
\]

\[
\text{if} \quad T(\mathbf{m}, \mathbf{x}) \neq d(\mathbf{x}) \text{ then}
\]

\[
J \leftarrow G(\mathbf{m}, \mathbf{x})(2d(\mathbf{x}) - 1) - b
\]

\[
\mathbf{m} \leftarrow \mathbf{m} - (2c_1/\sqrt{n}) J(2d(\mathbf{x}) - 1)(\mathbf{m} - \mathbf{x})/||\mathbf{m} - \mathbf{x}||
\]

\[
G(\mathbf{m}, \mathbf{x}) \leftarrow ||\mathbf{m} - \mathbf{x}||/\sqrt{n} - t
\]

\[
T(\mathbf{m}, \mathbf{x}) \leftarrow G(\mathbf{m}, \mathbf{x}) \geq 0
\]

\[
\text{if} \quad T(\mathbf{m}, \mathbf{x}) \neq d(\mathbf{x}) \text{ then}
\]

\[
J \leftarrow G(\mathbf{m}, \mathbf{x})(2d(\mathbf{x}) - 1) - b
\]

\[
b \leftarrow b + 2c_2 J
\]

\[
\text{endif}
\]

\[
\text{endif}
\]

Fig. 4. Modification procedure for membership vector of an FTU.
where $c_1$ and $c_2$ are learning constants. Equations 11 and 12 must be true for all input vectors $x^{(k)}$ and corresponding desired outputs $d(x^{(k)})$ yielding $p$ sets of equations for $p$ input vectors or training elements each with its own value of $b$, i.e. $b^{(k)}$. Note that $k$ labels a training element.

Parameters need only be adjusted for a particular $k$ if the following holds:

$$T(m, x^{(k)}) \neq d(x^{(k)}).$$

This is a less stringent condition than expression 7 and is utilised in the algorithm, shown in Figs. 3 and 4.

### 5. Recurrent Network of Fuzzy Neurons

A fully connected recurrent network, shown in Fig. 5, consisting of the FTU’s described in the previous section may be used to represent various mappings based on transformation from state to state. The dynamics of such a network called FRANN (Fuzzy Recurrent Artificial Neural Network) is such that the network changes state synchronously with all units changing states simultaneously. Input to the network consists of putting the network in some state called the starting state, and output of a value is the state of the network at some future point in time which may be after one or many transitions depending on the mapping that is desired.

The state transition function, $T(x)$, for the $i$th unit is given by:

$$x_i \leftarrow (G(M_i, x) \geq 0),$$

where $G(M_i, x) \geq 0$ is the binary valued function for the binary fuzzy unit. $M_i$, the $i$th row of a membership matrix $M$, is the membership vector for $(0, 0, \ldots, 0)$ at the $i$th unit. Since updating is synchronous, one transition or state change, $T(x)$, for the whole network may be described using vector notation, by:

$$x \leftarrow (G(M, x) \geq 0).$$

$T(x)$ is now a vector valued function of a vector and $G(M, x)$ is a vector whose $i$th component is $G(M_i, x)$. In case of synchronous updating, a state transition is made up of the $n$ individual transformations performed by $n$ FTU’s in parallel. A neural network operating in a synchronous mode can therefore be readily simulated via parallel processing computers, and its algorithms can be easily expressed in an array processing language such as APL or J.

If the transformation, $T$, is applied repeatedly, a cycle will be reached eventually since there are only a finite number of possible states. If there is a sequence of states $x^{(0)}, x^{(1)}, \ldots, x^{(k)}$ such that:

$$x^{(i)} = T(x^{(i-1)}) \quad i = 1, 2, \ldots, k \quad \text{and} \quad x^{(0)} = x^{(k)}$$

and $x^{(i)} \neq x^{(i)}$ for $i < k$,

then the sequence is called a $k$-state attractor or a limit cycle of $k$. The superscript denotes time rather than a particular training pattern. The word attractor also denotes a relation between a state and a sequence of states that forms a cycle. If $y = T^q(x)$ for some $q$, and $y$ is part of a $k$-state attractor, then the $k$-state attractor is called an attractor of state $x$. An attractor from this point onwards will be assumed to be a one-state attractor. $a$ is called a one-state attractor or fixed point if $T(a) = a$. If $x$ is attracted to a one-state attractor, this attractor, state will be represented by $AT(x)$. Thus $AT$ corresponds to the mapping from states to one-state attractors using $T$, the required number of times to reach a fixed point.

$$AT(x) \equiv T^k(x),$$

where $k$ is such that $T^{k+1}(x) = T^k(x)$. (17)
The domain of $AT$ includes only attractees, which is the name chosen for the states that are attracted to attractors. States attracted to themselves (i.e. $AT(a) = a$) must be attractors and vice versa according to Lemma 1 in the Appendix.

6. Training and Growing of FRANN
FRANN is trained to incorporate attractors by adjusting the membership vector for each unit using the method of Sec. 4 with some modifications. The dynamics of the network in case of synchronous updating are such that all units change state simultaneously during a single change in state. The output of any unit in this case depends only on the input vector and the membership vector for that unit. Therefore for a synchronous one-transition mapping, $T(x)$, the units can be trained independently of each other and the training algorithm for one unit is simply extended to the whole network. The training cycle can be expressed by the nested loops: (for several epochs (for all patterns (for all units update $M$))). Thus feedback is provided on the percent of the patterns which map correctly through all units at the end of each epoch. The training algorithm is shown in Fig. 6. The input patterns correspond to $k$ while the units correspond to $i$.

At the end of each epoch, the percent correct may be calculated. If it exceeds the best so far, the membership matrix corresponding to the best so far is replaced. Gallant\textsuperscript{58,59} refers to this strategy as the pocket algorithm.

Growth of the network is effected if we start with a membership matrix all of whose components are equal. This means that classification is attempted with only one free parameter initially. All units are equivalent and the membership values for all input lines to a unit are also equal. We could also start with a membership matrix in which all rows are identical. In either case, all the units in the network are equivalent and there is only one unique unit. During training, the number of unique rows increases which means that the number of unique FTU’s increases. Growing takes place as part of this training method if we consider that the size of a network is not the number of FTU’s in total, but the number of unique fuzzy units. Thus the network grows in the number of unique fuzzy units required. The importance of growth during training is that no more than the required number of unique units is generated which allows for improved generalisation during classification. As we will see in a later section, the amount of processing required also depends on the number of unique units and is therefore reduced.

Generally, the number of unique units produced by this training method which allows growth, is small in comparison to $n$ which is the dimension of the state vector. Thus in some simulations the value of $n$ was 100 while the number of unique neurons was only 3.

7. Utilizing Redundancy in the Membership Matrix during Retrieval
Since there are generally far fewer unique units than $n$ generated during growing of FRANN, it is possible to reduce the number of calculations required for state transformations. Note that $n$ is the number of components in the state vector. Fuzzy membership matrix $M$, because of row redundancy, can be expressed as $M = R[r; c]$, where $R$ is the matrix of say $p$ unique rows and $q$ columns of $M$, while $r$ and $c$ are $n$ dimensional index vectors if $M$ is $n \times n$. The number of unique rows is the number of unique units. Thus $M$ can be represented by the triplet $(R, r, c)$. The number of elements for the triplet is

\begin{verbatim}
while desired accuracy has not yet been achieved, do the following for one or more epochs:
    for $k$ in random permutation of 0, 1, ..., $p - 1$
        for $i < n$
            Modify $m$ and $b$($M_i, b_i^{(k)}, x^{(k)}, M_i, b_i^{(k)})$
        endfor 
    endfor 
endwhile
\end{verbatim}

Fig. 6. Training algorithm for FRANN.
pq + 2n as opposed to \( n^2 \) for \( \mathbf{M} \). The degree of redundancy tends to be very large. If \( p \) and \( q \) are 3 and 4 respectively, and \( n \) is 100, the difference in the number of elements between the compressed and uncompressed form of \( \mathbf{M} \) is 212 versus 10,000.

Not only does using the compressed form of the membership matrix save on storage space, the compression also leads to great savings in calculations required to retrieve a state as the following shows. The derivation makes use of notation and lemmas provided in the appendix.

A new state is calculated according to the logic vector expression:

\[
\|\mathbf{M} - \mathbf{x}^T\| \geq 0.5\sqrt{n}. \tag{18}
\]

\( \mathbf{M} - \mathbf{x}^T \) is a matrix obtained by subtracting the vector \( \mathbf{x} \) from each row of \( \mathbf{M} \). \( \|\mathbf{M} - \mathbf{x}^T\| \) is obtained by finding the euclidian norm of each row of \( \mathbf{M} - \mathbf{x}^T \).

The expression above is equivalent to the binary expression:

\[
(\mathbf{M} - \mathbf{x}^T)^2.1 \geq 0.25 \ n. \tag{19}
\]

Squaring a matrix here means to take the square of each component. \( \mathbf{1} \) is the vector of all ones. The value of this expression is a binary vector with the same number of elements as the number of rows of \( \mathbf{M} \). The expression above is equivalent to:

\[
\mathbf{M}^2.\mathbf{1} - 2\mathbf{M}.\mathbf{x} + \mathbf{x}^T \mathbf{x} \geq 0.25 \ n. \tag{20}
\]

By adding \( \mathbf{1}.\mathbf{x} - \mathbf{1}.\mathbf{x} \) and because \( (\mathbf{1} - \mathbf{x}).\mathbf{x} = 0 \), an equivalent binary expression is:

\[
(2\mathbf{M} - \mathbf{1}).\mathbf{x} \leq \mathbf{M}^2.\mathbf{1} - 0.25 \ n. \tag{21}
\]

\( \mathbf{M}^2.\mathbf{1} \) is the dot product of the square of \( \mathbf{M} \) and a vector of all ones which is equivalent to adding the squares of the elements in each row of \( \mathbf{M} \). Because there are equivalent rows and columns in \( \mathbf{M} \), \( \mathbf{M} \) can be written as:

\[
\mathbf{M} = \mathbf{R} [\mathbf{r}; \mathbf{c}],
\]

where \( \mathbf{R} \) is the nub of \( \mathbf{M} \) which includes only the unique rows and unique columns of \( \mathbf{M} \). Then \( \mathbf{M}^2.\mathbf{1} \), by Lemma 8 in the appendix, can be expressed as:

\[
\mathbf{M}^2.\mathbf{1} = (\mathbf{R}^2.\mathbf{f}(\mathbf{c}))[\mathbf{r}]. \tag{22}
\]

Note that \( \mathbf{R}^2.\mathbf{f}(\mathbf{c}) \) is a matrix with one column. \( \mathbf{f}(\mathbf{c}) \) is a vector-valued function of a vector such that its \( i \)th component is the number of times the value \( i \) appears in \( \mathbf{c} \).

By Lemma 7, \( (2\mathbf{M} - \mathbf{1}).\mathbf{x} \) can be written as:

\[
((2\mathbf{R} - \mathbf{1}).\mathbf{f}(\mathbf{x} \# \mathbf{c}))[\mathbf{r}], \tag{23}
\]

where \( \mathbf{x} \# \mathbf{c} \) is the vector obtained by selecting from \( \mathbf{c} \) all the elements for which there is a 1 in the corresponding position in vector \( \mathbf{x} \).

Thus the binary transform for retrieval of a state can be written as:

\[
((2\mathbf{R} - \mathbf{1}).\mathbf{f}(\mathbf{x} \# \mathbf{c})) \leq (\mathbf{R}^2.\mathbf{f}(\mathbf{c})) - 0.25 \ n)[\mathbf{r}] \hspace{1cm} (24)
\]

The resulting binary transformation for determining the next state during retrieval expressed by equation (24) can also be expressed as:

\[
\mathbf{z} \equiv \mathbf{f}(\mathbf{x} \# \mathbf{c}),
\]

\[
(||\mathbf{R} - \mathbf{z}|| \geq \sqrt{0.25 \ n + \mathbf{z}.(\mathbf{z} - \mathbf{1}) - \mathbf{a})}[\mathbf{r}], \tag{25}
\]

where \( \mathbf{a} \equiv \mathbf{R}^2.\mathbf{f}(\mathbf{c} - \mathbf{1}) \) is a property of the network only and therefore has to be calculated only once. The left-hand part of the inequality has to be calculated only once and may be interpreted as a threshold vector of \( p \) components. The transform of a state \( \mathbf{x} \) is found by evaluating the binary expression:

\[
(2\mathbf{R} - \mathbf{1}).\mathbf{f}(\mathbf{x} \# \mathbf{c}) \leq (\mathbf{R}^2.\mathbf{f}(\mathbf{c})) - 0.25 \ n, \tag{23}
\]

and then by expanding the result by using the selection vector \( \mathbf{r} \).

This demonstrates that all that is required for transforming a state is to apply the nub of the membership matrix to a reduced state and then expand the result using the row selection vector for the nub of the membership matrix. The dot product \( (2\mathbf{R} - \mathbf{1}).\mathbf{f}(\mathbf{x} \# \mathbf{c}) \) typically requires far fewer calculations than \( \mathbf{M}.\mathbf{x} \). For example in some of the simulations, \( \mathbf{M} \) was found to be of dimension 100 by 100 while \( \mathbf{R} \) was of dimension 4 by 4.

The proceeding has shown that time complexity and space complexity may be greatly reduced by utilising the sparseness of the membership matrix.

8. **FRANN’s as Classifiers**

An attractor network such as FRANN may be used for classification if prototypes are stored as attractors. A prototype for a class is the ideal version of all patterns in the class while all other elements
are noisy versions of the prototype. An attractor by definition is such that \( T(a) = a \) and by Lemma 1, \( AT(a) = a \). If \( AT(x) \) exhibits the accretive property, then:

\[
AT(x) = y \Rightarrow AT(x + \delta) = y.
\]

Which says that if \( x \) maps to \( y \) under \( AT \), then perturbations of \( x \) should also map to \( y \). Thus noisy versions of prototypes should map to prototypes under \( AT \). This can be used in classifying an unlabelled element by initialising the network to the state representing the unlabelled element and determining its attractor. If its attractor is a prototype, the unlabelled element is assigned the class of that prototype. If it has no attractor or if its attractor is not a prototype, then it is considered to be unclassifiable by the network. i.e.

\[
\text{unclassified element } \rightarrow \text{prototype } \rightarrow \text{class identifier}.
\] (24)

The transformation from prototype to class identifier is represented outside the network by a look-up table. It is important to split the mapping from pattern to class identifier in two, so that the transformation from test element to attractor will exhibit the accretive property and the transformation from attractor to class identifier will be nonaccretive.\(^6\)

The table look-up is not included in training and its domain is constant. If something produced by the \( AT \) mapping is not in the domain of the table, it means that the original input is nonclassifiable. Effectively all attractors not included in the set of prototypes are nondecision states. This means that spurious attractors\(^6\) play a useful role in deciding whether a pattern is nonclassifiable.

The mapping is an auto-associative mapping and the network is trained so those prototypes become one-state attractors during training. Note that \( T \) can be replaced by \( AT \) according to the Lemma 1.

9. FRANN for Classification of Feature Vectors

A feature vector is a vector of values such that each value corresponds to some property of the object represented by the vector. We consider the case where the values are elements of \( \mathbb{R} \). If FRANN is to be used for classification of feature vectors, real values have to be converted to binary values. This is done using the algorithm in Fig. 7. Thus if the field width is 4, the number 3 would be represented as 1101. The vector \((2, 1, 3)\) would be presented as 1100100111, assuming field widths of 4, 3 and 3.

For each feature

Convert feature value to a nonnegative integer using the formula

\[
k \leftarrow \text{int}((v - l)w/(u - l) + 0.5).
\]

(COMMENT \( l \) is the minimum value for the selected feature based on training data
\( u \) is the maximum value for the selected feature based on training data
\( v \) is the value to be converted
\( w \) is the number of positions in the binary representation for the selected feature
\text{int} is the symbol for “nearest integer less than or equal to.”)

Replace the integer by \( k \)'s followed by \( w - k \) 0's
Append binary pattern to previous binary pattern for preceding feature endfor

Fig. 7. Algorithm for converting feature vectors to binary vectors.
Note that it would seem obvious that a real value should be represented by its base 2 equivalent for the sake of compactness. The problem with this is that then, within a binary field corresponding to a feature value, positions differ in importance because of positional notation.

Note also that part of the training for classification includes the selection of a field width for each feature value. Should each feature be allowed the same field width or should the field width depend on the range to be represented? This author feels that the field width should be dependent upon the relative importance in terms of classification of the field to be represented. Since this is generally not known beforehand, it is felt that each feature should be accorded the same field width representation. The correct field width to be used for all the feature values can only be determined by experimentation. It would seem best to start with small widths of say one and work upwards. A small field width would not allow enough characteristics of the data to be represented, while a large field width may not allow generalisation.

10. Simulations

The simulation values for the components of $M$ are initially set at 0.5 which is maximum fuzziness. A fixed value of 0.5 for threshold, $t$, was found to be satisfactory although learning of $t$ was originally included in the algorithm. Small positive values of 0.01 are selected for $b^{(k)}$, where $k = 0, 1, \ldots, l - 1$. $b^{(k)}$ is never allowed to become negative. If $\Delta h^{(k)}$ is such that the new $b^{(k)}$ is less than 0, $\Delta h^{(k)}$ is set to 0. Each row of $M$ must always be $\in [0, 1]^n$. The learning rates $c_1$ and $c_2$ are set at 0.1. Learning speed is measured in terms of epochs required to reach a level of correctness.

The method was tried on several sets of training data for which the results of applying other methods have been reported. One such data set is the Australian credit dataset (2 classes, 14 attributes, 690 observations) described in Michie. The purpose of classification in this case is to develop a rule for validating applications for credit cards. The results of 21 different methods are reported in Michie, giving a range of classification accuracy in testing from 79.3% to 86.9%. The median was 84.8%. Back propagation according to Michie produced classification accuracy of 84.6%.

In case of application of FRANN, the results were as shown in Table 1. The values for the learning rates, $c_1$ and $c_2$ were set to 0.1 for all simulations. 382 elements were tested for class 1 and 306 elements for class 2. Training was accomplished by storing a prototype for each class as a fixed point. The prototype was obtained by taking the average of all the vectors in a class. Various sized state vectors were used to represent feature vector. All training and test states were attracted to fixed states. A state attracted to a nonprototype attractor is counted as not being classified. Training required no more than 2 epochs. Note that only 3 unique FTU’s are required for the best classification accuracy of 87.2%. This may be compared to the best classification accuracy reported by Michie of 86.9%.

A second data set consists of a sugar diabetes data base (2 classes, 8 attributes, 768 observations). The classification accuracy reported in Michie in this case ranges from 67.6% to 77.3%.

<table>
<thead>
<tr>
<th>Dimension of State Vector</th>
<th>Percent Correct</th>
<th>Percent Unclassified</th>
<th># Epochs for Training</th>
<th>No. of Unique FTU's</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>71.7</td>
<td>13.4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>85.8</td>
<td>14.5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>42</td>
<td>87.2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>70</td>
<td>87.2</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>140</td>
<td>84.3</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>210</td>
<td>85.0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>280</td>
<td>82.8</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>700</td>
<td>83.0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1. Australian credit classification using FRANN.
Table 2. Sugar diabetes data classification using FRANN.

<table>
<thead>
<tr>
<th>Dimension of State Vector</th>
<th>Percent Correct</th>
<th>Percent Unclassified</th>
<th># Epochs for Training</th>
<th>No. of Unique FTU's</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>62</td>
<td>1.2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>63.1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>54.5</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>72.8</td>
<td>0</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>58.2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>48</td>
<td>51.4</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>56</td>
<td>50.0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>63</td>
<td>48.0</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>80</td>
<td>65.1</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>160</td>
<td>46.5</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3. Iris data classification using FRANN.

<table>
<thead>
<tr>
<th>Dimension of State Vector</th>
<th>Percent Correct</th>
<th>Percent Unclassified</th>
<th># Epochs for Training</th>
<th>No. of Unique FTU's</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>68.7</td>
<td>6.25</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>76.6</td>
<td>6.94</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>77.1</td>
<td>11.84</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>86.8</td>
<td>0.7</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>91.0</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>32</td>
<td>81.2</td>
<td>6.9</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>40</td>
<td>67.4</td>
<td>27.1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>80</td>
<td>60.4</td>
<td>30.6</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

According to Michie, back propagation produced a classification accuracy of 75.2%. In Table 2 are the results obtained using FRANN. The number of components in the state vector is the field width times the number of fields which is 8. The number of test elements is 499 for class 1 and 267 elements for class 2. FRANN grows to only 3 unique FTU’s. The result of a classification accuracy of 72.8% obtained with FRANN is better than the results obtained by the majority of the methods reported in Michie.

A third data set consists of the well-known Iris data set. There are 3 classes of Iris which are Setosa, Versicolor and Virginica. The 4 features which are used are petal width and length, and sepal width and length. There are 50 feature vectors available for each class. This is a three-way classification rather than a two-way classification.

11. Summary

It has been demonstrated how a fuzzy recurrent artificial neural network consisting of fuzzy binary units can be grown for the purpose of classifying feature vectors or other patterns. A fuzzy unit classifies its input by determining a membership value for the input in the fuzzy set associated with the unit. Inputs whose membership value exceeds a threshold or cut value belong in one category while the others belongs in the other category. Training of the recurrent network is done by training individual units. The training algorithm for an individual unit is based on an algorithm for linear inequalities developed by Ho and Kashyap. Hassoun uses an algorithm also based on this method in Ho-Kashyap recording. Simulations have demonstrated that the proposed transform is effective as a discriminant function and that the proposed training algorithm can be used to
store fixed points in a fully recurrent network for the purpose of classification. The advantages of the network and its training method are: (1) The training time is extremely short; (2) Recurrent networks such as this one are generally readily implemented in hardware; (3) The size of the network is generally much smaller than the size of the network which would be obtained if the algorithm did not consist of growing. The reduced size implies better generalisation, smaller storage requirements and fewer calculations. In fact, the classification accuracy obtained on several standard data sets with this method is better than that obtained by the majority of other standard methods; and (4) The use of fuzzy logic is very intuitive since class membership is generally fuzzy.

Further work may consists of incorporating the field width (the number of bits used to represent a feature value) as a direct parameter in the training algorithm and allowing each feature to have its own field width rather than having the same field width for each feature.

Acknowledgments

A Scholarly Activity Grant provided by the University College of the Cariboo has partly supported this work.

Appendix

Nomenclature

- The origin for subscripts is 0.
- Matrices are represented by bold capital letters, such as $A$, $B$, and $M$.
- Vectors are always assumed to be column vectors unless otherwise specified, and they are represented by lowercase boldface letters such as $a$, $b$, and $x$.
- To simplify the expressions, which follow, we adopt the notation developed by Iverson.\textsuperscript{54}
  - Square brackets are used for subscripts. Thus if $a$ is a vector, $a[i]$ denotes the $i$th component. If $A$ is a matrix then $A[i; j]$ denotes the element of $A$ in the $i$th row and $j$th column.
  - $A[i;]$ denotes the $i$th row of $A$, and $A[; j]$ denotes the $j$th column of $A$. The blank after the semicolon indicates that all columns are selected while the blank before the semicolon indicates that all rows are selected.
- $A - b^T$ is the matrix obtained by subtracting $b$ from each row of $A$.
- $\|A\|$ is obtained by finding the Euclidian norm of each row of $A$.
- The binary operator $[\ ]$ is extended so that the subscript may also be a vector of indices. Thus if $r \in \{0, 1, \ldots, k - 1\}^n$, then $a[r]$ is the $n$-dimensional vector obtained from $a$ by selecting from $a$ using the appropriate indices, i.e.

\[
(a[r])[i] = a[r[i]] \quad i = 0, 1, \ldots, n - 1.
\]

Example:

If $a = (0.3, 0.4, 2.6)^T$ and $r = (0, 0, 1, 0, 1, 2, 0, 2)^T$, then $a[r] = (0.3, 0.3, 0.4, 0.3, 0.4, 2.6, 0.3, 2.6)^T$.

When $r$ is used in this role, we will call it a selection vector. Note that $k$ cannot be greater than the dimension of $a$.

- The binary operation $A[r; c]$ is a function of a matrix $A$ and an index vector $r \in \{0, 1, \ldots, n - 1\}^c$, where $n$ is the number of rows of $A$. The result of the operation is also a matrix but with dimensions $r \times m$ where $m$ is the number of columns of $A$ and $r$ is the number of elements in $r$. The $i$th row of the new matrix is the $r_i$th row of $A$.
- The binary operation $A[; c]$ is a function of a matrix $A$ and an index vector $c \in \{0, 1, \ldots, m - 1\}^c$, where $m$ is the number of columns of $A$. The result of the operation is also a matrix but with dimensions $n \times c$, where $n$ is the number of rows of $A$ and $c$ is the number of elements in $c$. The $j$th column of the new matrix is the $c_j$th column of $A$.

Note that both index vectors $r$ and $c$ may be of any length.

- The function $A[r; c]$ is a function of a matrix $A$ with dimensions $n \times m$ and index vectors $r$ and $c$ such that $r \in \{0, 1, \ldots, n - 1\}^r$ and $c \in \{0, 1, \ldots, m - 1\}^c$. The result, $B$, returned by the operation is also a matrix but with dimensions $r \times c$. It is defined as:

\[
B[i; j] = A[r[i]; c[j]].
\]

For example, if

\[
A = \begin{pmatrix}
-0.5 & 0.6 & 0.1 \\
0.8 & -0.9 & 0.7
\end{pmatrix},
\]
Each column of \( K \) selects a component of \( a \) represented in a compressed form as several duplicate rows and/or columns, it can be obtained later in the appendix.

The usefulness of all this is that if a matrix \( B \) has several duplicate rows and/or columns, it can be represented in a compressed form as \( A[r;c] \). Not only are the storage requirements less, the calculations may also be reduced by using the nub of \( B \).

**Lemmas**

Following are Lemmas used in the report. The following Lemma is required for Sec. 8 of the report.

**Lemma 1**

\[
\forall x \ A_T(x) = x \Leftrightarrow T(x) = x .
\]

**Proof**

\[
T(x) = x \Rightarrow T^n(x) = x \Rightarrow A_T(x) = x
\]

\[
A_T(x) = x \Rightarrow x \text{ is a fixed point} \Rightarrow T(x) = x
\]

The following Lemmas are required for Sec. 7 of the report.

Selection can be replaced by post multiplication by a matrix as the following shows. This post multiplication permits other useful results to be readily obtained later in the appendix.

**Lemma 2**

\( a[c] = a.K \) and therefore \( a[c].x = a.K.x \)

Where \( K \) is such that \( K[i; j] = \delta[i; c[j]] \), \( \delta \) is the Kronecker delta function (i.e. \( K[i; j] = 1 \Leftrightarrow c[j] = i \)).

The dimensions of \( K \) are \( p \times q \), where \( p \) is the number of components in \( a \) and \( q \) is the number of components in \( c \).

**Proof**

\[
(a[c])[j] = a[c[j]]
\]

\[
(a.K)[j] = a.K[j] = \Sigma_i a[i]K[i; j] = \Sigma_i a[i]\delta[i; c[j]] = a[c[j]]
\]

Each column of \( K \) has exactly one 1 with the rest of the elements in the column being all 0’s. Each column selects a component of \( a \).

Example:

If \( c = (0, 0, 1, 0, 1, 2, 0, 2) \), then \( K \) is

\[
\begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]

**Definition**

Frequency distribution \( f(v) \) is a vector valued function of a vector of non-negative integers, \( v \), such that \( f(v)[i] \) is the number of times that \( i \) appears in \( v \).
Lemma 3
If \( K \) is the binary matrix derived from selection vector \( c \) as per Lemma 2 and \( 1 \) is a vector of all 1’s then \( K.1 = f(c) \)

Proof
\[
K[i:]1 = \Sigma_j \delta[i;c[j]] = (f(c))[i]
\]

Definition
The binary operator, \(#\), is defined as follows:
\( a \# b \), where \( a \) is binary, is a vector obtained from \( b \) by selecting those components for which there is a 1 in the corresponding position in \( a \).
For example \((1, 0, 1, 0) \# (3, 4, 5, 1) = (3, 5)\).

Lemma 4
If \( K \) is the binary matrix derived from selection vector \( c \) as per Lemma 2 and \( x \) is a binary vector then \( K.x = f(x \# c) \).

Proof
\[
(K.x)[i] = K[i:]x = \Sigma_j \delta(i;c[j])x[j] = \Sigma_j((c[j] = i) \text{ and } (x[j] = 1)) = (f(x \# c))[i]
\]

Example:
Let
\[
x = (0, 1, 1, 0, 1, 0, 0),
c = (0, 0, 1, 0, 1, 2, 0, 2)^T.
\]
Then
\[
K = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]
\( K.x = (1, 2, 1) \).
And also
\[
x \# c = (0, 1, 1, 2)^T,
f(x \# c) = (1, 2, 1).
\]

Lemma 6
\( A[r; c].x = A.K.x = A.f(x \# c) \)

Proof
Follows from Lemmas 2 and 4

Lemma 7
\( A[r; c].x = (A.K.x)[r] = (A.f(x \# c))[r] \)

Proof
This is a combination of Lemmas 5 and 6.

Lemma 8
\( A[r; c].1 = (A.K.1)[r] = (A.f(c))[r] \)

Proof
Follows from Lemma 7 and \( f(1 \# c) = f(c) \)
References


44. S. Y. Kung 1993, Digital Neural Networks (Prentice Hall, NJ).


