Auto-associative memory using n-tuple techniques

by J.M.Bishop, R.J.Mitchell

The use of n-tuple or weightless neural networks as pattern recognition devices has been well documented [1]. They have a significant advantages over more common networks paradigms, such as the multi-layer perceptron, in that they can be easily implemented in digital hardware using standard random access memories. To date n-tuple networks have predominantly been used as fast pattern classification devices. This paper describes how n-tuple techniques can be used in the hardware implementation of a general auto-associative network.

1 Introduction

Over the last ten years the field of neural networks has re-emerged and there is now considerable work in the area all over the world. The subject is variously called connectionism, parallel distributed processing, connection science and neural computing. These are all synonymous and define a mode of computing which seeks to include the style of computing used in the brain: “neural computing is the study of networks of adaptable nodes which, through a process of learning from past examples, store experiential knowledge and make it available for use” [2].

Neural networks consist of simple processing elements (neurons), suitably interconnected, which can learn from data presented to them. What makes them powerful is their potential for performance improvement over time as they acquire more knowledge about a problem, and their ability to handle fuzzy real world data. That is, a network taught certain data patterns is able to recognise both these data patterns and those which are similar; the network can generalise. Also, they can operate in parallel, and hence can be fast.

Neural networks can be used for such diverse purposes as control, instrumentation, image processing, speech processing and economic forecasting. The three basic types of neural network used to perform these functions are: predictive networks, neural classifiers and pattern associators. This paper is concerned with the latter function, a task most humans perform with ease.

Associative learning is employed whenever patterns are stored such that they can be re-evoked in the future; usually by the application of a fuzzy, or partially complete, trigger pattern. The most common form of associative network is the Hopfield network [3].

The paper describes the standard n-tuple method [1] and the innovations developed by the authors which have led to the design of an n-tuple auto-associative memory.

In the following section the basic n-tuple technique is reviewed. This is followed by a description of how n-tuple techniques can be used to implement a simple binary auto-associator. Subsequent sections introduce an extension to the basic n-tuple sampling method enabling it to be used on continuous data, and this leads into the
A practical system requires many such neurons and these neurons form a memory called a Class Discriminator. The data can be either exclusively sampled, such that each value is used only once, or oversampled by a factor of $m$, where each value is used $m$ times.

To learn a pattern, the $n$ boolean values from each sample are used to form an $n$-bit address into an array (or RAM), where there is one such RAM per $n$-tuple sample, and a boolean TRUE value is stored at this location (see figure 2 which depicts a RAM being used to learn a three bit tuple).

To classify an image, the data are sampled as before. However, instead of storing a TRUE value at each defined location, a count of the TRUE values stored at these addresses is maintained. This count is the system response to an unknown pattern. A block diagram of the system is shown in figure 3.

If the input pattern is the one taught, then the count would be 100%; a count of less than 100% indicates a lower degree of similarity to the training pattern.

2 The n-tuple method

In n-tuple networks, the basic model of a neuron is a standard random access memory or RAM. The inputs to the neuron are binary values which are used to form an address to the RAM; the output of the neuron is the data value stored at this address.

There are various methods by which n-tuple networks may be configured and used. The following shows how a simple group of such neurons can be used to classify binary images; however, more complex configurations with multiple layers or feedback are also possible [4, 5].

There are two stages to the operation of these networks. First they are taught appropriate data; this is the learn phase. Then they are used to classify new data; this is called the analyse phase.

2.1 The basic classification method

Consider the classification of the black/white image of figure 1; this is stored as an array of boolean values. This array is then sampled, according to some predefined schema (usually either randomly or sequentially), with $n$ such samples forming one $n$-tuple; in figure 1 below three samples are used to form one three bit tuple.

![Fig. 1 Selection of points from input image](image1.png)

![Fig. 2 Learning a 3-tuple](image2.png)

![Fig. 3 A class discriminator](image3.png)
In practice, however, many similar patterns are taught to the system, so each neuron is able to ‘learn’ different tuples. Therefore, when a pattern is shown to the system which is similar to, but not identical to, any pattern from the training set, the count may still be 100%, because, for example, the first tuple from the new pattern may be the same as the first tuple from the third pattern learnt, the second tuple may be the same as that from the sixth pattern, and so on. In general, the system is said to have recognised the object if the count exceeds p%, where p is chosen suitably, and is likely to depend on the application. Details on the amount of learning needed can be found in [6].

It is important to ensure that there are sufficient patterns in the training set so as to allow the system to generalise. However, the system can be taught too much data, in which case the memory becomes saturated. Then the system may recognise patterns which are too dissimilar from the training set. The saturated state can occur well before all the memory locations are used.

Practical systems often implement the n-tuple technique in a sequential mode; that is the first tuple is learnt, then the second, and so on. However, it is possible for the tuples to be learnt in parallel, the resulting system thereby operating much faster. Also, in a practical sequential implementation, a class discriminator need not consist of numerous small physical RAMs. Instead, one large RAM can be used, the lower address lines for which specify the tuple, the other lines specifying a virtual RAM.

It is worth noting that for these networks a given training pattern needs to be presented to the system only once. This is significant; for the multi-layer perceptron type network [7], a training pattern may need to be presented several hundred times, which can take a long time.

Also, the technique is independent of the types of input patterns presented to the system. Video images are often used, but the subject matter of these could be almost anything, for example they could be of products being sent down a production line, or of faces, etc. Other input data could be derived from audio, infra-red or ultrasonic signals; the method has been successfully used to classify mass spectra [8].

2.2 Discriminating multiple classes

The system described above could, for example, be trained using a set of images of one person’s face. Thereafter it could be shown an image of a face and report its similarity to that of the person taught. If the system were taught images of two or more people, then again the system could report if the face shown to it was one of those people. However it could not specifically identify the person whose face was presented.

The method described above can be extended easily so as to allow the system to both recognise different classes of object and discriminate between them. This is achieved by having many sets of RAM neurons, one for each class of image being learnt.

In action, the data patterns associated with each class are taught into the one discriminator associated with that class, in the manner described earlier. In analyse mode, the tuples are formed as before, and a count is made for each discriminator of how many tuples have been learnt. The object is then ‘recognised’ as the one whose discriminator ‘fired’ the most, that is, the discriminator with the highest count of memorised tuples. Two measures of confidence can be used [9], Absolute Confidence (AC), and Relative Confidence, (RC):

\[
AC = \frac{\text{Most Number of 'fires' - Next highest Number of Tuples}}{\text{Most Number of Tuples}}
\]

or

\[
RC = \frac{\text{Most Number of 'fires' - Next highest}}{\text{Most Number of 'fires'}}
\]

As stated earlier, discriminator memory is not filled even when adequate training has been achieved. Therefore it should be possible to improve the usage so as to reduce the amount of memory required. Appropriate techniques for this purpose are discussed elsewhere [10].

2.3 Discriminating similar classes

One problem with the n-tuple technique occurs when the data being discriminated are very similar. For example, consider a character recognition system which is trying to recognise and classify 8*8 images of characters and for which the tuple size is 8. Figure 4 shows two similar characters, for c and for e, for which there are 8 pixels which are different (shaded grey). These two images would be sampled in the manner described above with the c being ‘taught’ into one discriminator memory and the e ‘taught’ in another.
If, in the worst case, when sampling the input data the eight different pixels were sampled and formed into one tuple, then seven of the eight tuples used to sample the image will be identical. Therefore, if an \( e \) is presented to the two discriminators, all eight neurons in the \( 'e' \) discriminator will fire, and seven neurons in the \( 'c' \) discriminator will fire.

If, however, one sample of each tuple came from the area where the pixels differ, then all eight tuples will be different, so all eight neurons in the \( 'e' \) discriminator will fire, and no neuron in the \( 'c' \) discriminator will fire.

Thus the sequence in which the input is sampled and the tuples formed, \textit{the tuple mapping}, can have a significant effect when discriminating between similar classes.

Now consider the two letters \( i \) and \( l \). Here, there are also only a few pixels which differ, but the differing pixels are not in the same locations as those for the letters \( c \) and \( e \). Thus, if the discrimination between \( i \) and \( l \) is to be maximised, then a different tuple mapping is needed to that for the letters \( c \) and \( e \).

When the letters being analysed are very different, for example an \( E \) and an \( i \), then the system will have little difficulty discriminating between the classes, irrespective of the tuple mapping.

Hence, when attempting to discriminate between multiple classes, some of which are very similar; one tuple mapping is used on the widely differing classes, with other, customised mappings being used to discriminate between the similar classes.

How should a suitable tuple mapping be determined? One method which has proved effective is to use genetic algorithms [11]. A genetic algorithm is a general purpose search technique [13] which operates in the following manner.

The system maintains a number of suggested solutions to the problem; this is called a population. Associated with each solution is a measure of the performance of the solution. At each iteration of the algorithm, a number of new solutions are ‘bred’ from the current population and the performance of each solution is identified. The algorithm repeats until a good enough solution is found, or a certain number of iterations has occurred.

The choice of parents from which new solutions are bred is determined by the performance measures; better solutions have an increased chance of being selected. Good parents should yield good children, thus good solutions should be able to produce better solutions.

When applying the technique to optimum discrimination in n-tuple networks, the suggested solutions are the tuple mappings, and the performance measure is the Relative Confidence, \( RC \), as defined above.

In more detail, the members of the population contain a string specifying the addresses in the input data from where each tuple sample is taken. Initially, these addresses are set randomly such that each location in the input data is sampled. New mappings are found by swapping some of these addresses, so that the tuples are sampled from different locations; however, the system will still sample all of the input data. Early in the operation of the search algorithm, many of the addresses are swapped, so that rapid traversal of the search-space is achieved; this ensures that the system is not trapped in a local maximum. However, as the algorithm progresses, fewer addresses are swapped so that the technique can converge onto the global maximum.

This method has been successfully applied to the problem of character recognition [12]. First groups of similar characters are identified and then the genetic algorithm is applied to each of these groups (for example \( i \) and \( l \), as well as \( c \) and \( e \)) to determine the appropriate tuple mapping; for other characters a normal random mapping is appropriate. The complete system, therefore, consists of many discriminators, many of which use one tuple mapping, but others use specific mappings. On the tests performed on characters (as detailed in [12]), the technique increased by 10% the number of recognised tuples.

3 Auto-associative n-tuple networks

The above describes one use of n-tuple networks, that is for pattern recognition / classification. More complex configurations enable the use of an auto-
An auto-associative network should be capable of processing an input, identifying that input as belonging to a particular class, and then producing an output which is the archetype, that is, the perfect version, of that class. For example, suppose the system were trained on images of faces of three people, then the three classes would be those three people. If the system was shown a noise affected image of the first person, then the system should produce a noise free image of that person. Such a network can be implemented in the following manner [5].

The input is sampled as above, and n-bit tuples formed, except the sampling process continues until each bit has been sampled n times; the input is said to be oversampled n times. This ensures that the output vector is the same size as the input vector.

Also, in learn mode, instead of writing a ‘1’ in the addressed location in each RAM neuron, the value written in the ith neuron is the binary value in the ith location of the class archetype. This is illustrated in figure 5 below.

![Image of storing an archetype vector](image)

**Fig. 5 Storing an Archetype vector**

In analyse mode, the system does not count the number of neurons which fire, instead each neuron outputs the value stored in the addressed location. The value output from the ith neuron will be, if the sampled input is the same as that taught, the ith value from the archetype.

As for normal n-tuple networks, the system is taught many similar inputs, so that it is able to generalise. That is, even if the input is not the same as one taught to the system, most of the values output will be the appropriate values from the archetype.

There is a complication, however, as it is possible to store many archetypes in the same memory. Identical tuples may be formed from two different inputs, so it is possible that when learning one archetype a ‘0’ should be stored in the neuron, but when learning another a ‘1’ should be stored at the same location.

To handle this problem, each location in the neuron is capable of storing two bits of data which are classified as being GROUND, CLASH, STATE0, or STATE1. Initially all locations are given the GROUND value. Then when learning, the following algorithm is used to process the value in the addressed location in memory (this is shown in figure 6, where the operation is on the third RAM):

```plaintext
IF (Value in neuron = GROUND) THEN
    Store data value from archetype into neuron;
ELSIF (Value in neuron <> data value from archetype) THEN
    Store CLASH in neuron;
```

![Image of learning an association](image)

**Fig. 6 Learning an association**

In analyse mode, the value output from the ith neuron should be a STATE0 or STATE1 value stored in the neuron at the address specified by the tuple. If, however, that location contains GROUND or CLASH, then the neuron has no valid data. The best guess for that value is the value in the ith location of the input data, so this is output. The associated algorithm is (see figure 7):
IF (Value In Neuron = STATE0 OR Value In Neuron = STATE1) THEN Output Value from Neuron; ELSE Output ith Value from Input;

The above is improved if feedback is connected around the system. First the input is presented to the system and a suitable output is produced which should be closer to the archetype than the input. Then this output is feedback and presented as the input and sampled in the same manner, producing an output which is even closer to the archetype. After a few iterations the system stabilises, producing an output close to the archetype of the class to which the input belongs. Figure 8 shows the initial response of such a system, trained on two archetype facial images, when presented with a distorted version of one as input, and the system response after 1, 2, 3, 6 and 12 feedback cycles.

It is possible to store many classes of data in one memory, whereas a standard system requires one set of memory for each class. For example, the authors reported in [13] that 10 classes of 8*8 characters could accurately be stored in one such auto-associative memory.

The tests associated with the processing of characters showed that various factors affected the performance of the system. The system was taught the archetype for each class and versions of the archetype with one bit distorted. The system was then shown versions of the archetype with many bits distorted and the success of the technique was determined by how close the output was to the associated archetype.

It was found that an 8-bit tuple size was better than a 6-bit tuple size; that the system responded better when it was taught with all versions of the input with one bit distorted than when, say, the system was taught a quarter of these inputs; and that the system responded better when there were fewer classes taught [13]. These observations make sense, as when more versions of the distorted images are shown, it is more likely that the tuples will address locations where data have been written. However, when more classes of data are taught, there is a greater chance of clashes. As regards tuple size, if the tuple size is too small the system cannot generalise; for many situations, experimentation has shown that a tuple size of 8 is appropriate.

4 Minchinton Cells

A standard n-tuple net cell consists of a RAM element with its address inputs defined by boolean states sampled from the input set. If this set is derived from an analogue source then some pre-processing is needed to convert the analogue values into the boolean states required to define the address input to the RAM. The simplest process is to threshold the input, sampling a true or false condition depending on the magnitude of the analogue input relative to a reference level.
However, a large amount of information is lost using this method, while the logical address input patterns to the RAM will vary greatly with changes of overall amplitude of the source.

A conventional solution to this problem is to use 1-in-n or Thermometer Coding [1]. This involves converting each sampled point of the input into an array of boolean values, where the greater the amplitude of the source, the more boolean values are true. This is equivalent to maintaining a number of threshold systems, each with a different threshold. This considerably increases the amount of memory required; for a $k$-bit analogue source the input vector size is increased by a factor of $2^k$.

The following section describes a novel form of processing of analogue sources into the boolean values suitable for input into an n-tuple RAM neuron. The method has a general form of which three specific examples will be discussed.

4.1 The Minchinton cell

A Minchinton is a processing element which is connected between the analogue source set and the address input to the RAM neuron. Each cell has inputs from a number of points in the analogue source set, the number of inputs being dependent upon the function required from the cell.

4.2 Examples of Minchinton Cells

If $V(x)$ is a randomly sampled value at location $x$ in the analogue source set $V$, the following three cell functions can be described:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V(x_1) &gt; \text{constant}$</td>
<td>Threshold</td>
</tr>
<tr>
<td>$V(x_1) &lt; V(x_2)$</td>
<td>Type 0</td>
</tr>
<tr>
<td>$[V(x_1) - V(x_1 + 1)] &gt; [V(x_2) - V(x_2 + 1)]$</td>
<td>Type 1</td>
</tr>
</tbody>
</table>

From the above it can be seen that thresholding is a particular form of the general technique, hence 1-in-n coding can also be implemented using Minchinton cells.

4.3 Invariance to a constant DC offset

The Type 0 cell can often be used instead of 1-in-n coding and has a number of potentially useful characteristics. For any particular sample, $V(x_1)$, if another sample, $V(x_2)$, is chosen randomly, the probability of the Cell giving a true output is given by:

$$\frac{\text{Number of points of amplitude} < V(x_1)}{\text{Total number of points in set}}$$

Thus the cell gives an output dependent upon the relative amplitude of a point $V(x_1)$ to the other points in the set, and not the absolute amplitude. If this is repeated over the complete sample set, the overall response will be independent of the mean value of the input set, provided that the input signal does not saturate.

As the output of the cell depends upon the actual random points chosen, a practical system will require a degree of oversampling. The amount of oversampling increases the ability of the system to discriminate and to take account of relative amplitude. The Type 0 cell, coupled to a stochastic search network, has been successfully used for feature extraction within grey level 2D images [14].

4.4 Invariance to zonal DC offsets

This type of cell simply compares two points in the analogue source set by calculating the difference between adjacent points, each pair being chosen randomly. The effect of the Type 1 cell is to render the network insensitive to zonal DC changes in the analogue input set, such as shadows, except at zonal boundaries. However, as the Type 1 cell involves a differential process, the network becomes more sensitive to high frequency noise, a problem that can be reduced by applying a low pass filter to the input.

The Minchinton cell thus provides a means of sampling from a continuous analogue source set to produce binary data suitable for input to RAM neurons. It can thus be used in a standard n-tuple recognition system. It has also been used as a component in the general n-tuple auto-associative memory, described below.

5 A general auto-associative memory

The authors have extended the binary associative network to handle continuous data, for example, grey levels from 0 to 255. Here, the neurons must be able to store the values 0..255 as well as GROUND and CLASH. This is described fully in [15].

For binary images, the memory needs two bits per pixel, in order to store the values GROUND, CLASH, STATE0 and STATE1. Here, assuming the images contain pixel values in the range
0..imax, then the memory needs to store the values 0..imax, GROUND and CLASH. The algorithms given earlier may be used, except that valid data are now in the range STATE0..STATEimax, not just STATE0 or STATE1.

A block diagram of the auto-associative network in learn mode is shown in figure 9. The archetype, placed in the archetype store, is sampled linearly; and the input image, held in the input store, is sampled randomly using a Type 0 Minchinton cell, so as to form the tuples. The values written into the neurons are those obtained from the archetype store.

![Fig. 9 Learning in an associative memory](image)

In analyse mode, the input image is sampled as for the binary system, but using Minchinton Cells to convert the analogue source data into binary data with which to form valid addresses to each RAM neuron, and the data stored at these address locations are used to generate the output image. If the value in the RAM is GROUND or CLASH, then the corresponding value from the input image is returned (as for the binary associator), otherwise the value in the RAM (in the range 0..imax) is returned.

As can be done for binary data, feedback can be used to improve the performance of the system. Here the output image is copied back into the input store and the sampling process repeated. After a few iterations, the system reaches a stable state with the output image being closer to its archetype than the image input. The system was tested using two classes of 256 level images of faces of size 32*32. Figure 10 shows the output of the system when processing distorted versions of these classes; in both cases the output images have fewer erroneous locations than their input images. The factors which affect the performance are, as for the binary case, the tuple size, the amount of learning and the number of classes taught.

![Fig. 10 Example general associative recall](image)

6. Hardware implementation

The systems described above can easily be implemented in hardware. This is because the neurons are simple memories and binary data are passed between the components of the system, and these data can thus be processed by standard logic circuits. Indeed, the first commercial neural network system, called WISARD, was built over ten years ago [16].

More recently the authors have developed a modular n-tuple network system [17]. The modules required are connected in a pipe-lined system in appropriate configurations. A block diagram of an associative network constructed from such modules is given in figure 11.

More details about the implementation of these modules are described in [18], but as an example consider the system shown in figure 11. Here the modules include storage units whose sizes are configurable (up to 512 * 512, 1 to 8-bit resolution; sampling units for forming tuples (the tuple size can be set in the range 4 to 8); and the RAMs and associated data processing (they can just store a logic 1 in the addressed location, as is required for the standard WISARD type system or, for example, they can perform the algorithms required for the auto-associative network).
7 Conclusion

This paper has considered the use of n-tuple or weightless form of neural network as a general pattern associator. In so doing, the principles behind these networks have been demonstrated, some configurations of network discussed, and its hardware implementation considered.

Some simulation results are shown; illustrating the system converging on an image of a human face, given a distorted version of that image entered as input, for both the binary and the multi-level, auto-associative n-tuple pattern associator.

8 References