10 Applications of *N*-tuple sampling and genetic algorithms to speech recognition

A. Badii
Schlumberger Technologies, Central Research Department, Farnborough, UK

M. J. Binstead
17 Myddleton Road, Uxbridge, Middlesex, UK

Antonia J. Jones
Department of Computing, Imperial College of Science and Technology, University of London, London, UK

T. J. Stonham
Department of Electrical Engineering, Brunel University, Uxbridge, Middlesex, UK

Christine L. Valenzuela
Department of Computer Science, Teesside Polytechnic, Middlesbrough, Cleveland, UK

Abstract

N-tuple nets are conceptually a highly parallel architecture for pattern recognition, implemented in hardware as a device called WISARD. However, high-speed serial emulations of *N*-tuple nets offer considerable advantages of flexibility and cost efficiency in applications, such as speech recognition, requiring only moderate bandwidth.

In this chapter we first describe a software technique for designing dynamically evolved *N*-tuple nets and illustrate the process whereby the designed structure can be progressively mapped into hardware to a level determined by the application requirements.

Next, we summarize some simulation studies which apply *N*-tuple nets to isolated word recognition and vowel detection.

For isolated word recognition it is shown that with raw data (non-pre-emphasized, noisy speech), *N*-tuple recognition yields improvement over dynamic time warping, while providing substantial savings in processing time.

For vowel detection, two distinct, single-speaker studies are described.

In the first experiment we attempt to accommodate to variation in the length of articulation of a vowel by training six distinct discriminators for each class of vowel, each of the six being trained over a different timescale.

In the second experiment on vowel detection, results are presented for a task specific optimization of a single mapping WISARD pattern recognizer using Holland's genetic algorithm.

1. Introduction

In this chapter we provide a synopsis of work, carried out by the authors under the auspices of the Pattern Recognition Laboratory, Brunel University, on the application of the N-tuple sampling paradigm of Bledsoe & Browning[1] to speech recognition.*

Networks of the type under consideration are simulations of extremely stylized models of biological neural networks. Such systems are usually characterized by some very simple algorithm, frequently little more than an inner product, replicated a large number of times as parallel, sometimes loosely coupled, processes. Examples of such systems in the literature include perceptrons,[3] WISARD nets,[4] Kohonen's topologizing nets,[5] the goal seeking components of Barto & Sutton[6] and, more recently, the conformon nets of Fish.[7] In this chapter we will concentrate on the implementation of WISARD nets, described below, applied to speech recognition.

The advantages of the WISARD model for pattern recognition are:

- Implementation as a parallel, or serial, system in currently available hardware is inexpensive and simple.
- Given labelled samples of each recognition class, training times are very short.
- The time required by a trained system to classify an unknown pattern is very small and, in a parallel implementation, is independent of the number of classes.

The requirement for labelled samples of each class poses particular problems in speech recognition when dealing with units smaller than whole words; the extraction of samples by acoustic and visual inspection is a labour intensive and time consuming activity. It is here that paradigms such as Kohonen's topologizing network, as applied to speech by Tattershall, show particular promise. Of course, in such approaches there are other compensating problems; principally, after the network has been trained and produced a dimensionally reduced and feature-

* Section 2 is based on [2], and more detailed reports on the work described in Sections 3 and 4 will appear elsewhere.

clustered map of the pattern space, it is necessary to interpret this map in terms of output symbols useful to higher levels. One approach to this problem is to train an associative memory on the net output together with the associated symbol.

Applications of N-tuple sampling in hardware have been rather sparse, the commercial version of WISARD as a visual pattern recognition device able to operate at TV frame rates, being one of the few to date—another is the optical character recognizer developed by Binstead & Stonham. However, one can envisage a multitude of applications for such pattern recognition systems as their operation and advantages become more widely understood.

Typically the real-time system is preceded by a software simulation in which various parameters of the theoretical model are optimized for the particular application. We begin by describing a software framework which is sufficiently general to cope with a large class of such net-systems, while at the same time preserving a high degree of computational efficiency. In addition, the structure produced has the property that it is easily mapped into hardware to a level determined by the application requirements.

The rationale for believing that N-tuple techniques might be successfully applied to speech recognizers is briefly outlined by Tattershall & Johnson,[8] who demonstrated that N-tuple recognizers can be designed so that in training they derive an implicit map of the class conditional probabilities. Since the N-tuple scheme requires almost no computation it appears to be an attractive way of implementing a Bayesian classifier. In a real-time speech recognition system the pre-processed input data can be slid across the retina and the system tuned to respond to significant peaking of a class discriminator response, see Fig. 4.

Two types of application to speech recognition are discussed. First, comparative results for *isolated word, single-speaker speech recognition* are presented for a variety of N-tuple recognizers. These results are then contrasted with the observed performance for the same data using a standard dynamic time warping algorithm used as a control in this context.

Next, preliminary investigations in vowel detection are reported; two distinct experiments are described. These experiments were restricted to *vowel detection for a single speaker.* Both experiments used the same data. In the first experiment we attempt to accommodate to variation in the length of articulation of a vowel by training six distinct discriminators for each class of vowel, each of the six being trained over a different timescale. In the second experiment one mapping is used for all vowels, each vowel having a single discriminator, and Holland's genetic

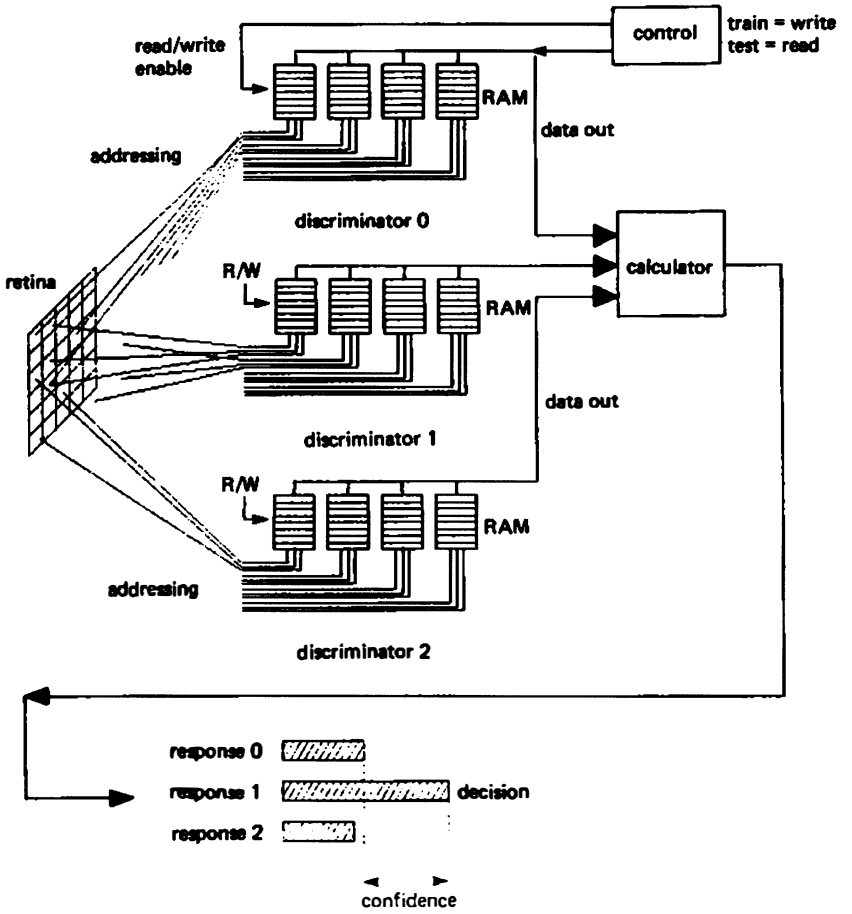


Figure 1 *Schematic of N-tuple recognizer.*

algorithm is used in an attempt to optimize this map for the specific task of vowel detection.

2. A simulation system

2.1 The WISARD model

WISARD (Wilkie, Stonham, Aleksander Recognition Device) is an implementation in hardware of the *N*-tuple sampling technique first described by Bledsoe & Browning.[1] The scheme outlined in Fig. 1 was first proposed by Aleksander & Stonham.[4]

The sample data to be recognized is stored as a two-dimensional array (the 'retina') of binary elements with successive samples in time stored in

successive columns and the value of the sample represented by a coding of the binary elements in each column. The particular coding used will generally depend on the application. One of several possible codings is to represent a sample feature value by a 'bar' of binary 1s, the length of the bar being proportional to the value of the sample feature.

Random connections are made onto the elements of the array, N such connections being grouped together to form an N-tuple which is used to address one random access memory (RAM) per discriminator. In this way a large number of RAMs are grouped together to form a class discriminator whose output or score is the sum of all its RAM's outputs. This configuration is repeated to give one discriminator for each class of pattern to be recognized. The RAM's implement logic functions which are set up during training; thus the method does not involve any direct storage of pattern data.

A random map from array elements to N-tuples is preferable in theory, since a systematic mapping is more likely to render the recognizer blind to distinct patterns having a systematic difference. Hard-wiring a random map in a totally parallel system makes fabrication infeasible at high resolutions. In many applications, systematic differences in input patterns of the type liable to pose problems with a non-random mapping are unlikely to occur since real data tends to be 'fuzzy' at the pixel level. However, the issue of randomly hard-wiring individual RAMs is somewhat academic since in most contexts a totally parallel system is not needed as its speed (independent of the number of classes and of the order of the access time of a memory element) would far exceed data input rates. At 512×512 resolution a semi-parallel structure is used where the mapping is 'soft' (ie achieved by pseudo-random addressing with parallel shift registers) and the processing within discriminators is serial but the discriminators themselves are operating in parallel. Using memory elements with an access time of 10^{-7} s, this gives a minimum operating time of around 70 ms, which once again is independent of the number of classes.

The system is trained using samples of patterns from each class. A pattern is fed into the retina array and a logical 1 is written into the RAMs of the discriminator associated with the class of this training pattern at the locations addressed by the N-tuples. This is repeated many times, typically 25–50 times, for each class.

In recognition mode, the unknown pattern is stored in the array and the RAMs of every discriminator put into READ mode. The input pattern then stimulates the logic functions in the discriminator network and an overall response is obtained by summing all the logical outputs. The pattern is then assigned to the class of the discriminator producing the highest score.

Where very high resolution image data is presented, as in visual imaging, this design lends itself to easy implementation in massively parallel hardware. However, even with visual images, experience tends to suggest that a very good recognition performance can often be obtained on relatively low resolution data. Hence in many applications, massively parallel hardware can be replaced by a fast serial processor and associated RAM, emulating the design in micro-coded software. This was the approach used by Binstead & Stonham in optical character recognition, with notable success. Such a system has the advantage of being able to make optimal use of available memory in applications where the N-tuple size, or the number of discriminators, may be required to vary.

2.2 The development of N-tuple systems

Practical N-tuple pattern recognition systems have developed from the original implementation of the hardware WISARD, which used regularly sized blocks of RAM that store only the discriminator states. As memory has become cheaper and processors faster, such heavily constrained systems are no longer appropriate for many applications. Algorithms can be implemented as serial emulations of parallel hardware and RAM can also be used to describe a more flexible structure.

In such a system we might require a dynamically variable number of classes, RAMs per class or mappings. N-tuple mappings need no longer map each retinal pixel uniquely and might be varied during training and across classes according to some heuristic supplied by the programmer—for example, Holland's genetic algorithm.[9] Having different mappings for each class does require that each class be given a separate opportunity to respond, but in some applications this may well be worth the extra overhead in time or hardware.

One might easily imagine that the price to be paid for this enhanced flexibility would be excessive complexity and slow performance. However, this turns out not to be the case and we will briefly outline why this is so.

2.3 Software system for dynamic reallocation of N-tuples

Conceptually it is helpful to think of the entire experimental design process of an N-tuple classifier as the growing and filling of a dynamic tree.

Initially this tree will have a root from which all else will grow. In practice 'root' is a pointer (down) to the first of the next level nodes, which for now we may choose to think of as class zero. (However, first-level nodes could equally be 'machine types' so that decomposition at the first level would then be into a series of parallel machines.) At the class level,

each class has a pointer (across) to the next class and a pointer (down) to the first RAM associated with that class.

We can iterate this process to create a tree–machine (ie data structure) which consists of:

(1) Classes—which in turn form collections of RAMs;

(2) RAMs—which form collections of input pointers (mappings) and pointers to the block of memory used to store the RAM state.

Fig. 2 illustrates the general structure of the tree. It is important to note that the nodes can hold extra information, for example statistics of their usage, a unique identifier and other pointers which can be used for memory control. This last feature is an essential part of a dynamically re-allocatable system.

Ultimately, memory will contain two types of information: the nodes which are joined by pointers to create the tree structure, and the memory which actually holds the taught information (the N-tuple storage). The memory requirement is strongly dependent on the N-tuple address size—adding an extra input to every RAM (although one could add an extra

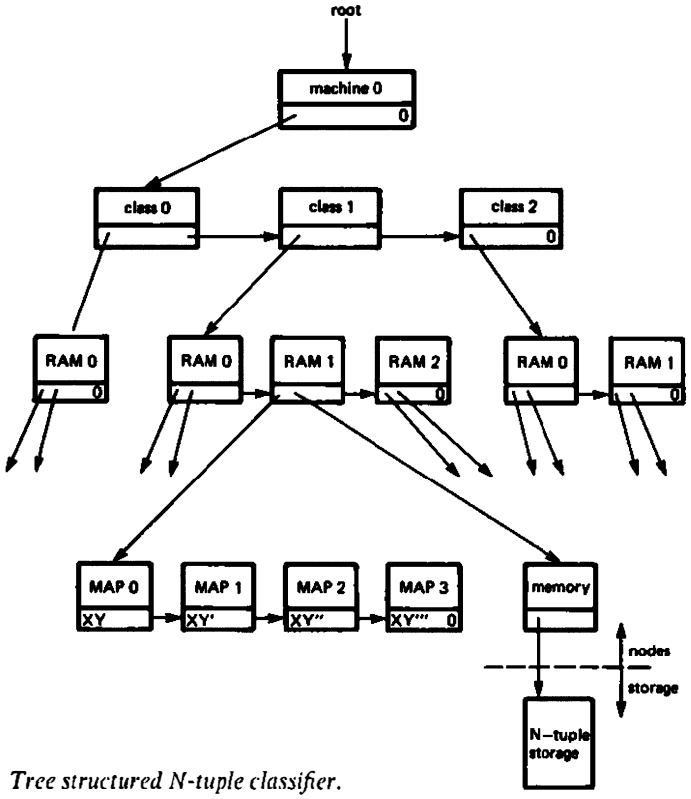


Figure 2 *Tree structured N-tuple classifier.*

input to just one RAM if desired) will linearly increase the number of nodes used but double the amount of N-tuple storage.

To access the memory it is necessary to traverse the tree to reach the requisite point. For example, suppose it was required to add an extra class. It becomes necessary to traverse the tree down to the class level and then along to the last-used class node, where a new node may be reclaimed from the 'node pool' maintained by memory control and added to form a new class by manipulating the necessary pointers. The same process may be repeated in order to add RAMs to the newly formed class.

In virtually every operation involving the tree a single very simple recursive algorithm, the *traverser*, is used. When calling the traverser, two parameters are passed: one is the base of the sub-tree to be traversed and the other is a pointer to a table of actions to be performed at each node visited. The table itself contains lists of actions for each possible node type. At present only two actions are used; the first is called when the node is entered and the other when the node is exited for the last time in the current traversal. For example, if one wanted to perform a classification: the first action on entering the node of type class would be to clear that class's response; upon leaving, the score (number of addressed RAMs in the discriminator which contain a logic '1') will have been updated by the lower levels so that the second action might be to print its value and to check if it is larger than the largest class score so far encountered.

Depending on the network being modelled the node types and actions can be chosen appropriately. For instance, if Kohonen's topologizing network were being modelled, one node type would be a *node*, in Kohonen's sense, which stores a state vector of the dimensionality of the data—his network is essentially an array of such nodes, and one action would be to modify the states of 'nearby' nodes according to the response of the current node to the data being presented.

A C-code listing of the traverser algorithm is given in Appendix I.[2] In most cases it will not be necessary to visit all nodes of the tree. So the traverser algorithm has extra switches that allow branches to be bypassed or the traversal aborted. In this way, for example, the search can be confined to a single level of the tree and aborted when a specific condition or node is attained.

Thus a flexible and simple experimental system, having all the proposed properties, has been created. It is now relatively straightforward for the experimenter to implement his chosen heuristics to control the evolution of the final system design. Moreover, since the structure consists largely of threaded pointers, very little calculation is required during the training and testing phases. Consequently, simulation times are considerably reduced.

Comparisons with earlier simulation systems, such as JAN, give an improvement of a factor between 2 and 4. Direct comparison is difficult since the earlier systems were so slow that they were modified to look only at input data which had changed, and they only dealt with regular sized discriminators, etc. If systems such as JAN had to deal with variable-sized discriminators then accessing a multi-dimensional array, say (class, RAM, element), could no longer be done using tables and would involve two multiplications and one addition, whereas in the present system access is via a pointer and involves no calculation.

When the fully trained system is complete the network of pointers will have become rather tangled. However, this poses no real problem since the structure of memory can be rationalized into appropriate blocks to facilitate implementation into hardware. This process is easily accomplished by a software module which reorders the pointers.

For historical reasons the final system has been named NEWJAM. It promises to be the vehicle for much of the net-systems research work of the adaptive systems and pattern recognition group at Brunel over the next few years.

2.4 Mapping the real time system into hardware

An important advantage conferred by NEWJAM is that since the data structure produced is tree-like it naturally decomposes into hardware at several alternative levels. Thus the actual decomposition can be chosen depending upon the bandwidth and response time required for the real-time system.

In Fig. 3 we sketch one possible approach for implementing the real-time recognition system (envisaged as a co-processor connected to a micro-computer host). The principal components of this system are:

68000/68020 CPU
This performs input–output functions and, initially, all actions called for via the action table memory. Every action is intrinsically a very simple process and consequently the most frequently called actions can be progressively replaced by special purpose hardware (Node type A processor, Node type B processor, etc., in Fig. 3).

Memory controller
This is the hardware which performs the traverser algorithm recursively. It could easily be implemented as a gate array and requires a small stack and access to a small number of status registers. In principle the traverser accesses the system memory via a separate bus (the tree bus) and can disable–enable the 68000 bus. In practice the traverser and the 68000 may share a common bus transparently, with the traverser able to control priority and refresh.

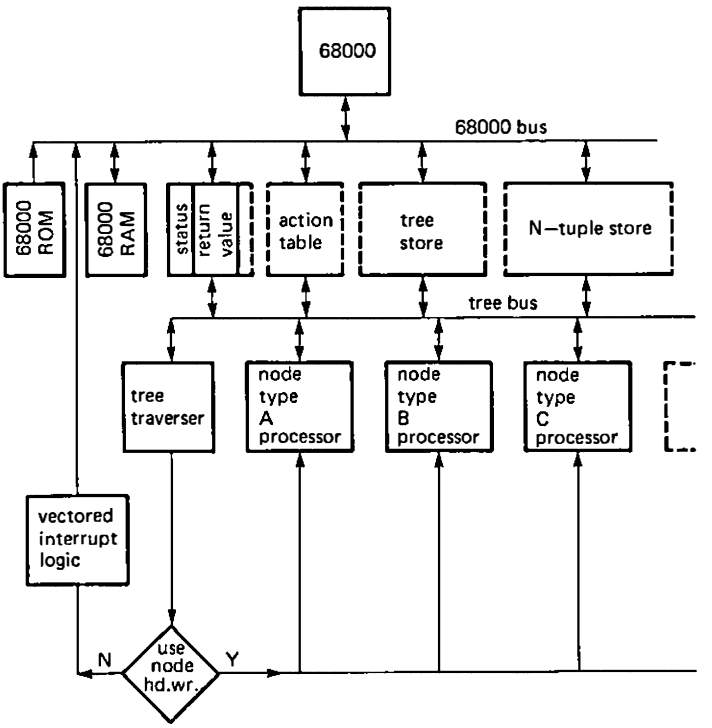


Figure 3 *Tree traverser—block diagram.*

Tree memory
The traverser locates a particular node of the tree by consulting a particular base address in tree memory. The block of memory starting at this address contains information describing the node (type, etc.). This memory is not particularly large and could be implemented in fast RAM.

Action table memory
Having located a particular node and recovered the address of the associated action type from tree memory the traverser consults this address in the action table memory which acts essentially as a function lookup table. As the number of action types is small this memory could be implemented in fast RAM.

N-tuple storage memory
This is the largest block of memory and can be implemented in slower, cheaper RAM.

When an action request is initiated, the corresponding module, or the 68000, must place an acknowledgement in the traverser status register.

Upon completion of the action a return value is placed in the status register.

Having decided upon the action type currently required the traverser places the request onto the action bus where it is either vectored to the 68000, if no special purpose hardware exists to perform the action, or passed to the appropriate action module. Initially there would be no action modules and the 68000 would perform all these actions. As action modules are slotted into the system they take over the corresponding role from the 68000.

An additional advantage conferred by this design is that if an action module fails, the 68000 can resume performance of the action until the module can be replaced.

3. Isolated word recognition

3.1 Introduction

In this section comparative results for *isolated word, single-speaker speech recognition* are presented for ten different N-tuple recognizers. These results are then contrasted with the observed performance for the same data using a standard dynamic time warping algorithm used as a control in this context.

Samples of 16 words from a diagnostic rhyming test list were collected from a single speaker on a carefully standardized data acquisition system (Shure SM12A microphone, flat pre-emphasis profile and a Sony model 701ES tape recorder) for subsequent automatic retrieval and digital processing using sample labelling and a modular A–D, D–A system with 16-bit resolution. This data was then stored on a VAX 11–750 to enable precise comparison of different recognition algorithms.

The speech data bank for the speech research includes the rhyming set, the alpha-numerics, simple command words and their synonyms, and the phonotactically permissible CVC–VCV constructs from a large speaker population under both controlled and noisy environments.

However, for the preliminary stages of the investigation it was decided to test N-tuple recognition systems under unfavourable signal conditions and using the minimum of pre-processing (ie non-pre-emphasized, non-normalized input speech). Thus if the performance of a simple system, operating on minimally pre-processed data from the rhyming set, was acceptable, then it could reasonably be expected that for a given corpus the early results would improve with a more advanced N-tuple recognizer using optimally tuned pre-processing and normalization techniques.

Accordingly, the experiments described here were run on data from the

noisy environment samples, allowing recognition to take place on sample data having no pre-emphasis or time normalization. Pre-processing was limited to a 19-channel vocoder bank,[10] simulated by fast Fourier transform (FFT), and scaling the result as input to the N-tuple recognizers.

The diagnostic test set was chosen so that the acoustic dissimilarity within rhyming sets (eg one/run—short) is minimal and the range of perceived phonological length did not markedly vary among the confusable rhyming sets (eg one/run/want—short; wonder/rudder—long). The 16-word diagnostic corpus was as follows:

Word set			
0	one	8	shoe
1	run	9	toot
2	want	10	tattoo
3	begun	11	toothache
4	wonder	12	cooler
5	rudder	13	tee
6	win	14	three
7	two	15	see

Two important dimensions of assessment for a speech recognition algorithm are: robustness in the face of a large speaker population and the rolloff in recognition accuracy as the vocabulary size increases. These aspects are *not* investigated in the present study, primarily because of resource constraints. However, this work represents a necessary first step in the evaluation of N-tuple sampling applied to speech recognition.

3.2 Experimental procedure for speech recognition

The strategy adopted for the present experiments was chosen to provide flexibility and repeatability with the same data, thus enabling comparison of differing recognition and pre-processing techniques. For this reason, simulations of the training and recognition process for eight different designs of N-tuple recognizer were performed on previously stored data using a VAX 11–750 system. Real-time performance was not a factor since it is known that the systems under consideration can be implemented with a satisfactory real time response when a suitable design has been proven.

3.3 Pre-processing algorithm

The raw-time domain files were subjected to a 10-ms wide FFT

producing 19 8-bit samples of each filter channel every 5 ms. In the first six experiments the 8-bit value was reduced to a 4-bit value using one of three encoding methods discussed below (encoding of data). The 4-bit intensity can be considered as a weighting of each pixel on the retina and the 19 samples as a single slice in time encoded as a vertical column on the WISARD retina. In this way each word was reduced to a 120×19 array of 4-bit elements. The total duration being 0.6 s.

After the first six experiments the 4-bit intensity of each filter channel was replaced by a single bit which was set if a pre-determined threshold (determined experimentally) was exceeded, thus reducing the word data to a 120×19 array of single bits for the final four experiments.

3.4 The WISARD retina

The WISARD retina was sized at 100 (horizontal) by 19 (vertical), each component consisting of four bits initially and one bit subsequently.

In the recognition stage of a real system the sample data can be visualized as stepping across the retina in steps of one horizontal unit (5 ms). Precise alignment in comparison with the training data would therefore not be a problem—as the data slid across, the system would be looking for a sharp peaking of one discriminator, see Fig. 4. Of course, one discriminator could be trained on the ambient noise. Thus segmentation of speech from background becomes an implicit property of this paradigm.

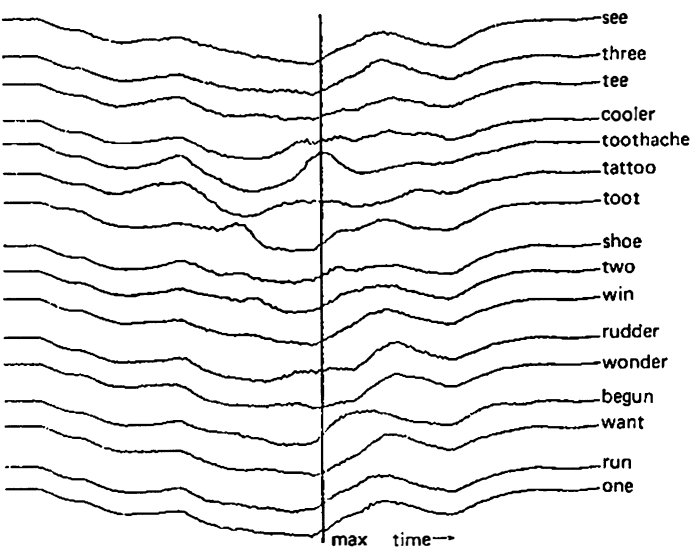


Figure 4 *Plot of all discriminator responses to 'toothache'.*

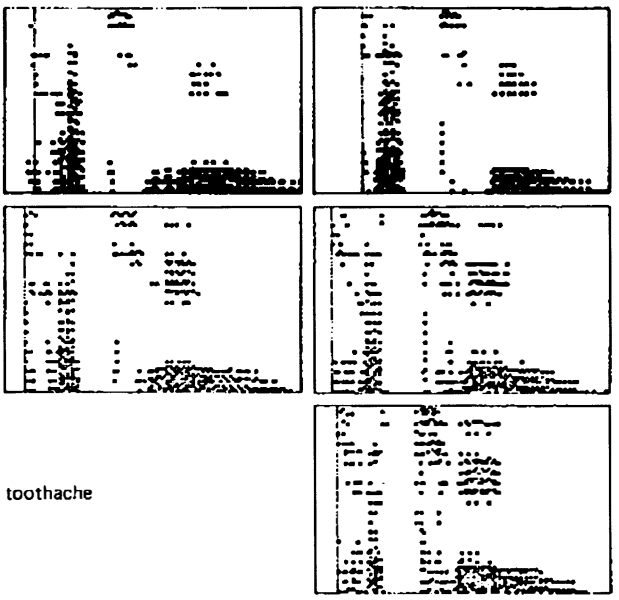


Figure 5 *FFT images of the word 'toothache'.*

Because the computational cost of scanning the image across the retina in 5-ms steps is too high in a simulation of this type, the start of a word in the sample frame was arbitrarily decided to occur when a 10% increase in the ambient energy level (summed across all filter channels) was observed. In training, each such sample was presented three times, representing a 'jitter' of ± 5 ms about the determined start point.

Fig. 5 shows FFT samples for the word 'toothache'. The vertical line indicates the time at which the threshold was exceeded; the subsequent 100 columns (500 ms) are taken as the retinal image.

3.5 Encoding and mapping

Four different kinds of encoding of the 8-bit samples produced by the FFT were employed. In the first six experiments each encoding reduced the 8-bit data to four bits. In the remaining two experiments the 8-bit sample was reduced to a single bit (binary encoding).

(1) *Linear-encoding*: here the top four bits of the 8-bit sample were selected and their binary image slotted into the retinal column in the position determined by which filter the output originated.

(2) *Thermometer-encoding*: for this encoding the interval [0, 255] was partitioned into five equal sub-intervals and integers in each sub-interval were mapped into a 4-bit value.

(3) *Gray-scale-encoding*: here the interval [0, 255] was divided into 16 equal sub-intervals. Each sub-interval is indexed by a 4-bit value in such a way that the Hamming distance between the indices of adjacent intervals is always 1. This form of indexing amounts to traversing all the vertices of a hypercube. The idea being that a small change in the value of the signal being encoded will produce a small change of Hamming distance in the encoded image.

(4) *Binary encoding*: finally the 8-bit sample was reduced to a single bit by thresholding at an experimentally determined level.

In the initial six experiments $N = 4$ and so $19 \times 100 \times 4/4$ N-tuples are chosen from the 1900×4 bits of the retina to define the mapping. Two types of mapping were used, namely *linear*, where N-tuple addresses are taken from consecutive pixels in a column, and *random*, where the addresses are composed from bits sampled randomly across the entire retina.

3.6 Results and conclusions for the 4-bit–4-tuple recognizers

Single-speaker recognition results with the 16-word repertoire. 4-tuple, 40-μs sampling rate (25 kHz, BW 0–8 kHz):

In the 4-bit encoding, 4-tuple experiments the best overall performance was obtained with linear encoding and a linear map or, equivalently, with Gray-scale encoding and a linear map. Initially we found this result rather unexpected in that the linear map employed took 4-tuple addresses from a single time slice, whereas the random map also looked across time. However, further comparison with the 1-bit encoding, 4-tuple experiments suggests that 4-bit encoding may have been presenting the system with excessive, relatively unrepeatable, detail.

It would appear that most learning occurs during the first five training instances of any given class, at which point the system gives around 85% accuracy. Subsequent training tends initially to reduce recognition performance and recovery is thereafter progressive but slow until saturation becomes a significant effect. We will return to the question of how the progress of the system towards saturation can be effectively monitored. However, our results suggest that with these system configurations, training on more than 25 instances from each class causes overall recognition performance to degrade.

With 4-bit encoding, a linear mapping and a 25-word teach set, the average performance of 90% looks quite promising as an initial result under the unfavourable conditions of the experiment. But the accuracy per word over the entire training sequence of 5, 10, 15, 20 and 25 patterns respectively was as shown in Table 1. Each discriminator consisted of 100×19 16-bit RAMs, ie a 3.8 (8-bit) Kbytes per word. Since there were

Table 1. *4-tuple–linear map–4 × 19 × 100–linear encoding*

Class	Training 5	10	15	20	25
			% Accuracy		
one	**60**	**36**	40	**44**	**52**
run	**80**	**76**	80	**76**	**88**
want	**48**	**72**	96	**100**	**96**
begun	**100**	**96**	96	**96**	**96**
wonder	**92**	**92**	88	**92**	**92**
rudder	**100**	**92**	92	**96**	**100**
win	**80**	**84**	80	**76**	**76**
two	**88**	**84**	84	**84**	**84**
shoe	**92**	**100**	100	**100**	**100**
toot	**92**	**88**	92	**92**	**92**
tattoo	**100**	**100**	100	**100**	**100**
toothache	**96**	**96**	96	**100**	**96**
cooler	**100**	**100**	100	**80**	**100**
tee	**80**	**84**	80	**100**	**80**
three	**92**	**96**	100	**76**	**96**
see	**60**	**44**	60	**88**	**92**
Average	**85.00**	**83.75**	86.50	**88.00**	**90.00**

16 class discriminators this comprised a total of 60.8 Kbytes of RAM used by the 4-bit–4-tuple recognizers.

Table 1 shows that the performance on the word 'one' (the worst case) was plainly unsatisfactory. A graphical confusion matrix for this experiment is given in Fig. 6. The confusion between the first three utterances, which uttered with no context would be particularly confusable even to the human listener, can mainly be ascribed to the fact that both the phonological duration as well as word-final and word-initial qualities are almost identical.

In an attempt to gauge the efficiency with which the discriminator RAMs were being used, two sets of statistics were produced for the case of 4-bit–4-tuple linear mapping with linear encoding. The first concerned the number of bits set in each 16-bit RAM versus class. The second gave the number of identical RAMs for all classes and the number of identical RAMs in pairs of classes. We briefly summarize this information.

Almost all zero-addressed locations were set, indicating that virtually every 4-tuple had seen (0, 0, 0, 0), ie a complete absence of activity in the retinal cells sampled, during training.

Typically, each discriminator had around 1000 ± 400 RAMs, from a possible 1900, with exactly one bit set. The previous observation suggests that in most of these it will be the zero-addressed bit which is set. So that

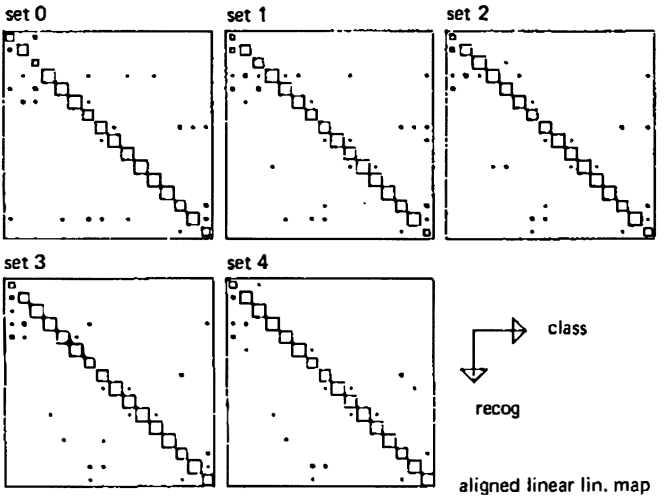


Figure 6 *Confusion matrices with 5/10/15/20/25 training examples.*

anywhere between 31 and 73% of the RAMs was each merely affirming the *absence* of some 16 particular activity features as a basis upon which to classify.

The number of RAMs per discriminator with more than one bit set was typically around 500. One might say that approximately 25% of RAMs were providing a contribution to classification based on between one and 15 observed activity features.

There were 91 RAMs which were identical for all classes. Thus most RAMs contributing on the basis of an observed activity feature were providing useful classification information.

Typically the number of identical RAMs in pairs of classes was in the range 500–1000, ie in any pairwise decision 50–75% of all relevant RAMs made a useful contribution, even if most of these were reporting absences of activity features.

Of the $28\,500 = 1900 \times 15$ non-zero-addressed bits per discriminator around 3000 were normally set (about 10%) as compared to a total number of bits set in the range 5000–7000 (max. possible 30 400). One can interpret this in one of two ways: one can argue that 10% RAM utilization is inefficient (in a 2-class system with ideal preprocessing the probability of any discriminator bit being set after training should be 0.5, with no commonality between discriminator contests); or one can say that this state of affairs reflects our ignorance of precisely what constitutes the critically significant features of the speech signal. (Such debates have a certain air of circularity.)

3.7 Results and conclusions for the 1-bit–*N*-tuple recognizers

A WISARD net is saturated when all discriminators give maximal response to sample data. This could occur, for example, as a result of over-training. In practice one trains the system almost to the point where the dynamic range of discriminator responses becomes insufficient to give an adequate margin upon which to base a classification decision.

To monitor the effectiveness of training in the last four experiments we define the following parameters of the system response with respect to any particular test sample:

$$\text{Response} = \begin{cases} \text{the discriminator score expressed as a percentage of} \\ \text{maximum possible.} \end{cases}$$

Min-response = the minimum response from any class.

Ave-response = the average response of all classes.

Let $D(i)$ be the response of the ith discriminator. For any particular class j let

$$d(j) = \max \{D(i): \text{all } i \text{ not equal to } j\}.$$

Thus $d(j)$ is the best response from all discriminators excluding the jth. Suppose now the data sample belonged to the jth class. Then $D(j)-d(j)$ is a measure of the margin by which the classification was made. If $D(j)-d(j)$ is negative then the sample was incorrectly classified.

Table 2. *4-tuple–linear-map–19 × 100–binary encoding*

Class	5	10	15	20	25
			% Accuracy		
one	96	72	52	76	64
run	56	64	64	76	76
want	88	92	96	96	96
begun	100	100	100	92	92
wonder	100	96	96	96	96
rudder	96	84	88	88	88
win	88	88	84	88	88
two	92	92	92	92	96
shoe	64	96	100	100	96
toot	92	100	100	100	100
tattoo	84	96	96	100	100
toothache	100	96	96	100	100
cooler	100	100	96	100	100
tee	76	72	80	84	84
three	92	96	100	96	96
see	92	92	96	96	100
Average	88.25	89.75	89.75	92.50	92.00

Table 3. *4-tuple–linear-map–19 × 100–binary encoding*

Class	5	10	15	20	25	Statistics
one	89.6	92.8	94.4	95.8	96.6	Response
	56.4	67.2	71.2	74.5	75.8	Min-response
	74.4	80.6	83.9	87.4	88.7	Ave-response
	2.3	0.7	−0.2	0.3	0.4	Margin
cooler	85.6	90.4	92.3	94.9	96.3	Response
	49.7	58.4	66.0	68.9	70.2	Min-response
	68.8	75.7	79.0	81.5	82.9	Ave-response
	7.9	6.8	6.2	6.0	6.1	Margin

As training and testing progresses, the quantity $D(j)-d(j)$ can be averaged over the test samples to provide a progressive picture of how training gradually reduces the margin of decision. Over a test set T of samples we can define for each class j:

$$Margin = \text{the average of } D(j)-d(j) \text{ over } T.$$

In the last four N-tuple experiments these statistics were collected to provide a running picture of the extent to which each class could benefit from further training.

Table 4. *4-tuple–random-map–19 × 100–binary encoding*

Class	5	10	15	20	25	% Accuracy
one	88	88	64	72	68	
run	52	72	56	72	76	
want	96	96	96	96	96	
begun	100	100	100	96	96	
wonder	100	96	96	96	100	
rudder	96	84	84	84	88	
win	92	92	88	92	88	
two	92	92	92	92	96	
shoe	60	96	96	96	96	
toot	92	92	92	92	92	
tattoo	64	88	100	100	100	
toothache	100	96	92	92	92	
cooler	100	84	88	96	100	
tee	76	56	48	72	68	
three	84	92	92	96	96	
see	92	92	100	100	100	
Average	86.50	88.50	86.50	90.25	90.75	

Table 5. *4-tuple–random-map–19 × 100–binary encoding*

Class	5	10	15	20	25	Statistics
one	90.4	93.8	95.3	96.3	96.9	Response
	31.4	40.6	48.2	59.4	61.9	Min-response
	63.2	70.6	75.0	80.9	82.9	Ave-response
	3.5	1.8	0.4	0.3	0.3	Margin
cooler	84.2	89.0	91.1	94.8	96.8	Response
	32.2	40.4	49.4	52.1	53.7	Min-response
	58.9	68.1	71.9	75.4	77.0	Ave-response
	10.4	7.0	6.2	5.7	6.5	Margin

The experiments were conducted for both 4-tuple and 8-tuple mappings over a wide range of threshold values (10- to 50-channel intensity). It was found that the systems were relatively insensitive to the threshold for the binary encoding over this range, there being almost no detectable difference in performance. We will present the results for a threshold of 20 as being typical in Table 2.

For the 1-bit–4-tuple recognizers the RAM cost is 950 bytes per discriminator, giving a total of 14.84 Kbytes for all 16 classes. However the margin of decision decreases very rapidly as training progresses. We give the worst and best case figures in Table 3.

Table 6. *8-tuple–linear-map–19 × 100–binary encoding*

Class	5	10	15	20	25
			% Accuracy		
one	92	88	80	84	72
run	64	76	84	88	84
want	88	96	96	92	92
begun	96	96	100	96	96
wonder	92	96	92	92	92
rudder	96	88	88	88	88
win	84	92	92	96	96
two	100	100	96	92	92
shoe	68	88	92	96	96
toot	92	92	100	100	100
tattoo	80	96	100	100	100
toothache	100	100	100	100	100
cooler	100	96	100	100	100
tee	76	60	76	84	84
three	92	96	100	100	100
see	72	76	88	96	96
Average	87.00	89.75	92.75	94.00	93.00

Table 7. *8-tuple–linear-map–19 × 100–binary encoding*

Class	5	10	15	20	25	Statistics
one	77.1	82.6	85.0	87.8	89.6	Response
	30.8	40.7	46.6	52.9	55.4	Min-response
	54.8	62.3	66.4	71.2	72.9	Ave-response
	3.8	3.2	1.9	2.4	2.6	Margin
cooler	67.0	74.5	78.5	82.7	86.3	Response
	25.3	32.1	38.3	40.3	41.3	Min-response
	45.9	52.6	56.3	59.4	61.2	Ave-response
	10.9	11.1	12.1	13.7	15.8	Margin

The result given in Table 3 is significantly better than the corresponding results for the 4-bit encoding experiments, at a fraction of the RAM cost. It provides evidence that the 4-bit systems were being presented with excessive detail. We next compare the corresponding performance with a random map (Tables 4 and 5).

Once again the linear map provides consistently better results. Turning now to the 1-bit-8-tuple results we have (Tables 6 and 7).

For the 1-bit–8-tuple recognizers the RAM cost is 7.42 Kbytes per discriminator, giving a total of 118.75 Kbytes for all 16 classes. The

Table 8. *8-tuple–random-map–19 × 100–binary encoding*

Class	5	10	15	20	25
			% Accuracy		
one	88	92	92	88	84
run	56	76	84	84	88
want	80	96	96	96	96
begun	100	100	100	100	96
wonder	96	96	96	96	96
rudder	96	88	88	88	88
win	92	96	96	96	96
two	92	96	92	96	96
shoe	52	76	92	96	96
toot	92	92	92	92	92
tattoo	56	88	96	100	100
toothache	100	100	100	100	100
cooler	100	88	88	92	100
tee	76	48	52	80	80
three	60	100	100	100	100
see	72	92	100	100	100
Average	81.75	89.00	91.50	94.00	94.25

Table 9. *8-tuple–random-map–19 × 100–binary encoding*

Class	5	10	15	20	25	Statistics
one	71.1	80.1	84.6	87.3	89.0	Response
	3.7	5.8	8.2	19.2	22.0	Min-response
	31.8	39.0	44.9	50.8	53.5	Ave-response
	7.5	7.1	4.3	4.1	3.3	Margin
cooler	55.3	66.1	70.4	76.2	82.4	Response
	4.4	7.1	11.0	12.0	12.9	Min-response
	23.8	31.6	35.5	38.7	40.7	Ave-response
	18.0	16.9	16.0	18.6	22.1	Margin

results are somewhat better and, as one might expect, the margin of decision decreases less rapidly as training progresses (Tables 8 and 9).

These final results are marginally better for the random map. This suggests that ability to perceive the logical conjunction of several formant features (in this instance an 8-tuple recognizer) is required before the expected advantage results from attempting to extract features across the time domain of a sliding FFT.

3.8 Comparative results using conventional time-warping

We next describe the results obtained with the original 16-word set but using conventional time-warping–template-matching recognition. Comparison of these results with those of the N-tuple recognition system shows that, on the same data, 8-tuple sampling provided significantly improved recognition accuracy.

3.8.1 DTW algorithm description

Assume, for the moment, that words are not finite temporally ordered sequences of spectra but continuously time-varying, vector valued functions. Suppose $\mathbf{a}(t)$, $\mathbf{b}(t)$ $(0 \leqslant t \leqslant T)$ are two words which we wish to compare. We may define a metric at the level of primitive patterns as

$$D(\mathbf{a}, \mathbf{b}) = \int_0^T d(\mathbf{a}(t), \mathbf{b}(t))\mathrm{d}t,$$

where d is some suitable metric of spectral difference.

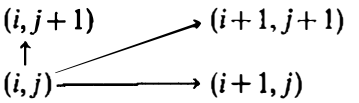
We know that very large local variations in the rate of articulation of a word can be tolerated without compromising its intelligibility. This suggests that a better metric should be largely invariant to changes of timescale. One way to accomplish this is to define a function $q(t)$ which maps the timescale of $\mathbf{b}(t)$ onto that of $\mathbf{a}(t)$. Modifying the previous

equation accordingly we obtain

$$D^*(\mathbf{a},\mathbf{b}) = \min_q \int_0^T d(\mathbf{a}(t),\mathbf{b}(q(t)))\mathrm{d}t.$$

Essentially this is an instance of a classical variational problem whose solution is found by solving the corresponding Euler–Lagrange equation. However, D^* must not be calculated with respect to an arbitrary change of timescale; we must place some constraints on q and these complicate the problem so as to make it, in general, analytically intractable. Fortunately, as Bellman has shown,[11] a numerical solution can be efficiently obtained by means of dynamic programming. It was this line of reasoning which first led Vintsyuk[12] to apply dynamic programming to speech recognition, often called dynamic time warping. The DTW algorithm described below is based on the work of Sakoe & Chiba.[13]

Let $\mathbf{a}_i$ $(1 \leqslant i \leqslant u)$, $\mathbf{b}_j$ $(1 \leqslant j \leqslant r)$ be sequences of spectral vectors. If $d(\mathbf{a}_i, \mathbf{b}_j)$ is a suitable measure of distance between $\mathbf{a}_i$ and $\mathbf{b}_j$ the DTW algorithm finds a path connecting $(1, 1)$ and (u, r) such that the cumulative distance is minimal, the guiding principle being that if a locally correct decision is made at every point then a globally correct path will be found (this is often obscured by specific implementations). If the current point is (i,j), then we choose the next point (i',j') by examining the three possible paths as illustrated below:



and choosing a path corresponding to the minimum value of

$$d(\mathbf{a}_i, \mathbf{b}_{j+1}), d(\mathbf{a}_{i+1}, \mathbf{b}_{j+1}), d(\mathbf{a}_{i+1}, \mathbf{b}_j),$$

where any point outside the rectangular region is omitted. The cumulative distance $D^*(i,j)$ is then updated:

$$D^*(i',j') = D^*(i,j) + d(\mathbf{a}_{i'}, \mathbf{b}_{j'}), \quad D^*(1,1) = d(\mathbf{a}_1, \mathbf{b}_1).$$

The final value $D^*(u,r)$ provides a time normalized measure of distance between **a** and **b**. When performing recognition the unknown **a** is compared against every **b** in the vocabulary and assigned the class for which D^* is minimal.

3.8.2 Results using conventional DTW

The DTW algorithm compares two arrays *ref* (the template—vertical axis) and *unknown* (the test sample—horizontal axis). Figs. 7 and 8 show complete cumulative distance contours and an optimal path for two runs

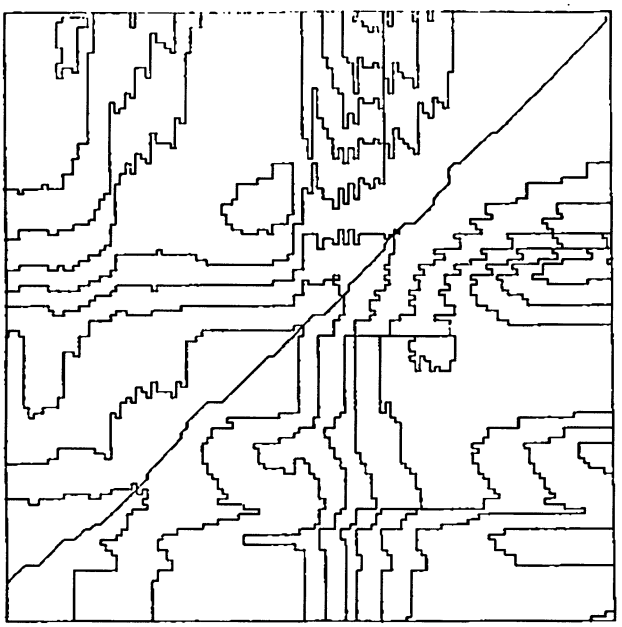


Figure 7 *DTW for two different sample of 'toothache'.*

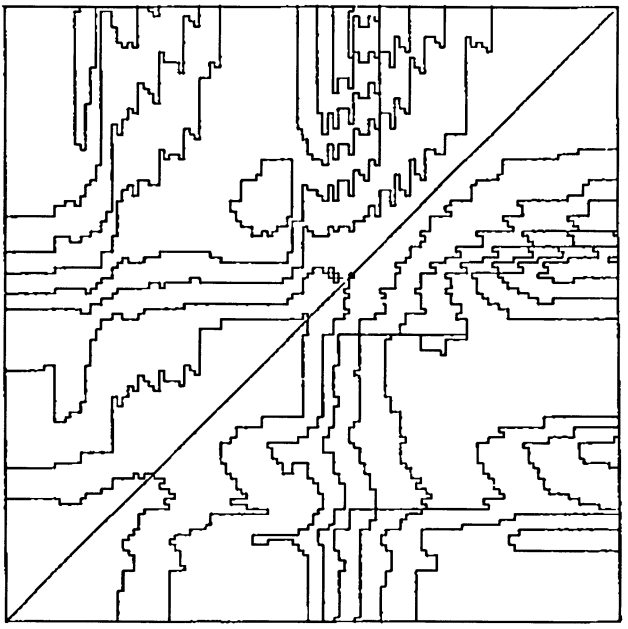


Figure 8 *DTW for time-aligned 'toothache' against reference.*

of the program. In Fig. 7 the word 'toothache' is compared with a different sample of the same word. As a test of these routines a sample of 'toothache' was compared with the reference 'toothache' and the resulting path used to warp the sample to conform to the reference. In Fig. 8 a second DTW is then performed, comparing the time-aligned sample against the reference; the resulting optimal path is, as expected, a straight line; this acts as a good test of the code.

In applying the algorithm, only one template is used for each reference word, but that reference is based on 5, 10, 15, 20 or 25 words taken from the teach sample. For example, in the first experiment five samples of the same word were selected. The first was taken as the basic reference and the remaining four were time normalized against the first in the usual way. In the sample vs. sample distance array so produced, each diagonal path was used as a time-distorting function to normalize the sample against the basic reference. Having eliminated as much time variation as possible all five samples were then averaged to produce the single reference.

It seems likely that one would get better results for DTW if each word in the teach set were used as a separate reference rather than by combining them as described above. However, the computational overhead in recognition would be so high that it is difficult to imagine a real-time system performing in this way.

Table 10. *DTW results*

Class	Training → 5	10	15	20	25
			% Accuracy		
one	76	80	84	84	84
run	80	88	88	88	88
want	92	92	92	92	92
begun	100	100	100	100	100
wonder	76	20	76	76	76
rudder	100	100	100	100	100
win	88	92	88	88	88
two	76	76	84	80	80
shoe	80	92	92	92	96
toot	80	80	80	80	80
tattoo	60	64	64	64	68
toothache	92	80	88	88	92
cooler	100	100	100	100	100
tee	92	92	92	92	92
three	92	92	92	92	92
see	100	100	100	100	100
Average	86.5	84.2	88.7	88.5	89.2

It would appear that the technique of averaging (time normalized) templates does provide some progressive improvement in accuracy as the number of templates increases—at least within the framework of this experiment—but that this improvement is not great (Table 10).

These are good results, admittedly at enormous computational cost, and emphasize the value of time normalization. Nevertheless, comparison with Table 8 shows that an 8-tuple WISARD recognizer (having no time normalization and, in principle, virtually zero computational overhead) obtained significantly better results on the same data. The inference would seem to be that if it were possible to provide a WISARD recognizer with time normalized data, at reasonable computational cost, the resulting system should have a remarkably good performance. This was confirmed by a later set of experiments.

3.9 Summary of results and conclusions

In this initial series of experiments in the application of N-tuple sampling to the problem of speech recognition some interesting lessons were learnt (Table 11).

These experiments demonstrate that under the most unfavourable conditions (noisy rhyming test utterances from a naive speaker, no pre-emphasis, no signal conditioning, no time or amplitude normalization) N-tuple sampling, applied to single-speaker isolated-word recognition with a 16-word diagnostic vocabulary, yields an improvement in accuracy of around 5% (in the range 90–100%) over conventional DTW using the same data.

Table 11. *Summary of results*

Tuple	Encoding	Data bits per channel	Mapping	RAM per word (bytes)	% Accuracy
4	Linear	4	Linear	3.8 K	90*
4	Linear	4	Random	3.8 K	88
4	Thermometer	4	Linear	3.8 K	79.75
4	Thermometer	4	Random	3.8 K	80.50
4	Gray	4	Linear	3.8 K	90*
4	Gray	4	Random	3.8 K	87.25
4	Binary	1	Linear	950	92
4	Binary	1	Random	950	90.75
8	Binary	1	Linear	7.42 K	93
8	Binary	1	Random	7.42 K	94.25
8-bit per channel – 19 channel DTW					89.20

* identical.

With amplitude normalization and active range encoding of the pattern vectors a further improvement can be expected to result.

Moreover, a WISARD implementation of N-tuple sampling has virtually no computational overhead (as compared to the high computational cost of DTW, or other recognition paradigms), and can, in principle, be built so that the response time is independent of the number of classes.

A further advantage of this paradigm is that for a real system discriminator responses monitored continuously can provide whole word recognition of connected speech without the necessity for segmentation.

4. Vowel detectors

4.1 Introduction

A desirable goal for a speech recognition system would be to identify phonemic segments of continuous speech accurately. Phonemic recognition need not be exceedingly accurate; accuracies around 80% might well suffice, since relatively simple linguistic knowledge based systems can detect something approaching 60% of randomly induced errors in a phonemic stream of English utterances (Badii, Hui & Jones—in preparation). Phonemic rule based error detection can also be enhanced to provide some degree of error correction. Higher levels of syntactic, semantic and contextual knowledge might then be used in a similar fashion to process the phonemic stream into text. Such a system could in principle cope with an unlimited vocabularly, in contrast to the limited vocabulary word recognition systems currently in use.

Certainly the goal of speech recognition must be beyond isolated word recognition towards the effective recognition of continuous speech. Systems such as COHORT and TRACE (see [14], Chapter 15, for example) point the way but do not promise cheap implementation in the medium run.

Despite the fact that some authors[15] report correct segmentation of continuous speech into phonemes with up to 97% accuracy, Rumelhart objects to segmentation before recognition:

> *Because of the overlap of successive phonemes, it is difficult, and we believe counterproductive, to try to divide the speech stream up into separate phonemes in advance of identifying the units. A number of other researchers (eg Fowler, 1984; Klatt, 1980) have made much the same point.* ([14], pp. 60–61)

Rumelhart prefers the approach of allowing the phoneme identification

process to examine the speech stream for characteristic patterns, without first segmenting the stream into separate units.

It is interesting that either approach is practical using a WISARD-type device. The advantage of prior segmentation is that it permits some degree of time normalization before presentation to the recognizer, and work at the Pattern Recognition Laboratory at Brunel University has shown that a very considerable improvement in recognition occurs if WISARD is presented with time-normalized data.

We may define a *static* pattern recognition system to be one which stores its training experiences in memory and refers to memory in seeking to classify unknown patterns. This contrasts with a *dynamic* system which continually undergoes state transitions and whose output depends on the current (and possibly previous) state(s) and the input rather than the input alone. While, dynamic pattern recognition systems are of considerable interest, the current theoretical situation is largely speculative and it seems likely that it will be some time before any practical system for vision or speech will be realized along these lines.

In a static pattern recognition system the goal is to optimize the map between input patterns and memory while preserving the real-time performance and keeping training to a minimum. In applications such as speech, the situation is rendered more difficult by the fact that the significant features of the signal are not really well understood. Without feedback, WISARD is a static model which makes no *a priori* assumptions about the input patterns and is easily implemented to give a suitable real-time performance.

As we have observed, WISARD is very simple and fast to train, provided one has suitably labelled samples of each class.

This last requirement creates serious logistical problems in applying static pattern recognition models to speech at a level below whole words. The speech signal must be examined visually and acoustically by a human operator who defines the boundaries of a segment which hopefully represents an example of the particular class. This sample can then be used for training or testing. Since many such samples are required for each class the construction of a suitable database is a very time consuming process. However, once such a database has been prepared it can be used for many different experiments and can enable direct comparison of different algorithms on identical data.

The experiments reported here were restricted to *vowel detection* for a *single speaker*.

4.2 Vowel detection using multiple discriminators per vowel

The words were pronounced in word pairs which instantiated the same

vowel in an attempt to obtain the coarticulative effects which would normally be present in continuous speech.

The sample speech was collected and passed through a 16-channel filter bank to produce frequency domain data. The frequency information was in 5-ms steps.

For both the training and test phases it was necessary to create a parallel file containing an indication at each step as to which class the 5 ms sample corresponded (or to no class). This second file was hand crafted and identification was accomplished by traversing the time domain data in small steps while playing back progressively nested samples through the D-to-A. Consequently, there is an element of subjectivity inherent in this identification process. One variant of each vowel was selected, these were:

A as in fAte
E as in mEt
I as in bIt
O as in gOat
U as in dUe

The *Concise Oxford English Dictionary* was used as a guide in defining which vowels were to be expected in the pronunciation of each word. It should be noted that various dictionaries are by no means in agreement as to the precise quality of each vowel that occurs in a given word and, of course, there is considerable variation between speakers.

In an attempt to deal with the fact that samples of a given class are liable to considerable variation of duration, each vowel segment in the training frequency data (once identified as above) was, in this initial experiment, linearly scaled to a uniform duration in order to fit a standard 16×16 8-tuple WISARD retina with one thresholded bit per pixel.

The variation in the vowel lengths was typically from 45 to 250 ms. Although we actually know how long the vowel samples are in the test phase, we cannot use this information during recognition, since the ability to cope with such variation is intrinsically part of the recognition process.

To deal with this we used six different scale factors. The incoming sound was placed in a buffer long enough to accommodate at least 250 ms (the longest observed vowel length). Every 5 ms this buffer was updated and six snapshots of differing lengths were presented to the WISARD recognizer. The classifiers with their different scale factors were treated as though they were separate classes so that during testing the highest responses would hopefully detect both the correct vowel and its duration.

Table 12. *Six discriminators for duration per class*

Vowel	Percentage recognition accuracy Duration and class	Class only
A	23.1	61.5
E	29.2	54.2
I	37.0	59.3
O	4.2	54.2
U	31.8	81.8
Average	25.2	61.8

4.2.1 Results and conclusions

As Table 12 might suggest, a confusion matrix for response against class and duration shows that correct classification of class was more reliable than correct classification of duration within the class. This is probably explained by the fact that estimating the vowel duration while preparing both the training and test data-classification file is a difficult and rather imprecise affair.

A second confusion matrix looking only at response against correct class is probably more significant and is given in Table 13. In general terms the idea is to present a sliding window of the frequency domain data from the test utterance to the WISARD net and determine whether the vowel discriminator responses are detecting the embedded vowels. Fig. 9 summarizes the result of one such simulation and consists of four traces. The top two traces indicate the strength of response and the confidence (the difference between the best and second-best classifications) for all window positions. The next trace details which vowel was producing the largest response as the words slid past the window. The bottom trace indicates where the vowel was found by the experimenter.

From these results it can be seen that single-speaker vowel detection from within continuous speech can be performed by a WISARD net using spectral energy data with a reasonable degree of accuracy.

Table 13. *Confusion matrix for class response*

		Classified as → A	E	I	O	U
Actual class	A	23	4	6	1	6
	E	3	14	7	2	—
	I	1	1	16	—	12
	O	20	—	1	15	1
	U	1	4	7	—	30

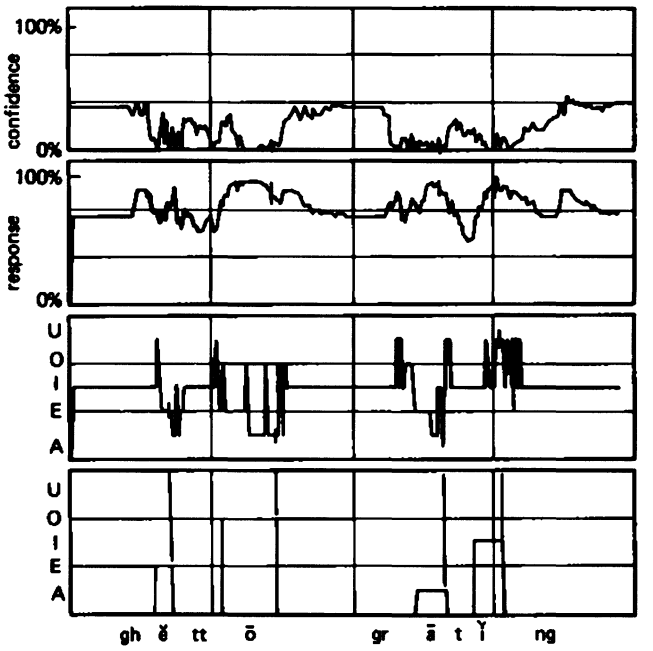


Figure 9 *Summary of continuous response.*

However, it should be emphasized that we are only attempting to recognize one particular type of each vowel quality.

It is possible to envisage a number of improvements in the experiment described above. For example, most of the energy in vowels is concentrated in the lower frequencies. Therefore a suitable pre-emphasis profile would no doubt improve the reliability of such a system.

The significance of these preliminary vowel detection results is to demonstrate the feasibility of using WISARD nets to recognize significant speech fragments within words of connected speech, but the results would be more interesting if generalized to a comprehensive set of building blocks such as phonemes or phoneme-like fragments.

4.3 Breeding vowel detectors using Holland's genetic algorithm

Given that the initial mapping from the retina to memory, that is, the assignment of N-tuple bits across the retina is random, the question arises as to whether the mapping can be improved for a particular type of application. For example, if the task were face recognition, then a better performance might be expected if the N-tuples were sampling more densely in that area of the retina where significant features such as the eyes, hairline, and mouth are presented.

As a vowel detector, a relatively difficult task, WISARD gives a reasonably creditable performance considering the lack of time normalization. Across the five classes, as we saw in the preceding section, typical recognition accuracies exceed 50%, and in particular classes are as high as 80%, as against the expected 20% of pure chance. Of course, carefully crafted vowel detectors can do much better than this, Jassem[16] reports accuracies of 92–97% in his review of speech recognition work in Poland. However, WISARD is a very simple recognition paradigm and the question addressed by the present experiment is: *by how much the*

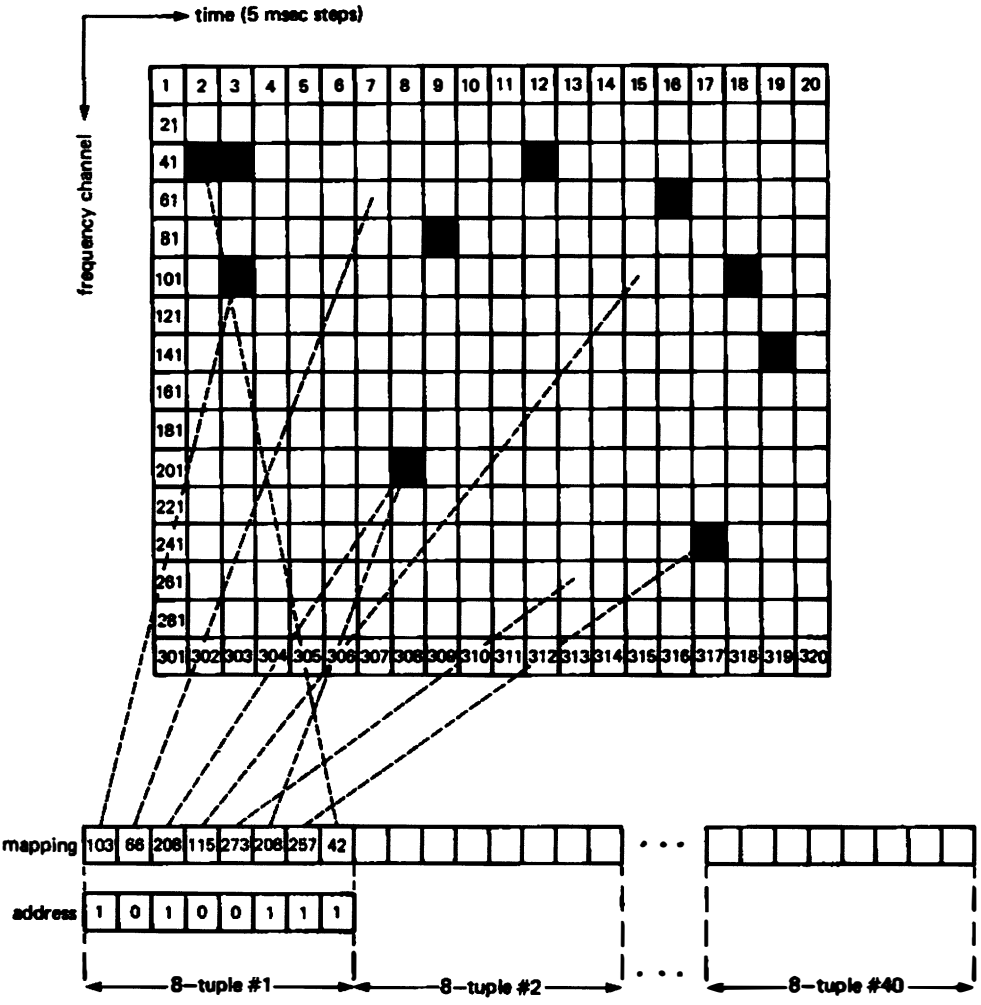


Figure 10 *Representing mappings as strings.*

performance can be improved using Holland's genetic algorithm to 'breed' better mappings?

Holland's algorithm[9] was chosen because it is a very powerful adaptive search technique and because the mapping from retina to N-tuples is easily described as a string: each position on the retina is numbered and each block of N such numbers in the string describes the mapping for a particular N-tuple, see Fig. 10. This is a particularly pleasant situation, because the usual difficulty with genetic algorithms is representing the objects being optimized as strings in such a way that after using a genetic operator, eg mutation to alter an element, the resulting string still represents a valid object. In the present case this is not a problem since any string of integers in the correct range (in this case [1, 320]) represents a valid mapping.

4.3.1 Holland's genetic algorithm

Simulated evolution had been tried before Holland with extremely poor results. All of these were based on the 'mutation and natural selection' model of evolution. Holland's genetic algorithms are based on a 'sexual reproduction and selection' model: his principal operator is crossing-over, that is, the creating of a new object for the next generation by combining parts from *two* independent objects in the current generation. Mutation plays a minor role in genetic algorithms.

Many experiments have been done with genetic algorithms, and they have proved to be remarkably effective and robust learning systems. For the most part they have been tested as function optimizers, where the objects in a generation are 'numbers' and their survival–reproductive value is given by the function whose maximum we wish to find.

One of the most interesting aspects of genetic algorithms is that they not only find the optimum object, but in doing so they discover properties that are common to many near-optimal objects (so-called higher-order schemata). In some instances, this information is at least as valuable as the optimum itself.

As the name 'genetic algorithm' suggests, the inspiration for Holland's work is taken from an analogy with biological systems. The mathematics of genetic evolution is now a very sophisticated tool which has changed our perception of how the evolutionary process works. For example, it is now known that simple mutation alone is insufficient to explain the rate of biological adaptation. Instead, mutation plays the role of background 'noise' which, by occasional random perturbation, prevents a specie from becoming frozen at a local optimum. Other factors explain the rapid rate of adaptation.

Holland constructs adaptive plan programs based on the following basic ideas. We are given a set, A, of 'structures' which we can think of in

the first instance as being a set of strings of fixed length, l say. The object of the adaptive search is to find a structure which performs well in terms of a measure of performance:

$$v: A \rightarrow \text{real numbers} \geqslant 0.$$

We have so far a knowledge base of competing structures and measure v of the observed performance of generated structures. For example, if the problem were one of function optimization the structures, or strings, could be the binary expansion of a real number to some fixed number of places, and the function v could be the function to be maximized. Then v evaluated at the real number represented by a string would be a measure of the string's fitness to survive.

Representing strings as

$$a(1)a(2)a(3)\ldots a(l) \quad (a(i) = 1 \text{ or } 0),$$

we can designate sub-sets of A which have attributes in common, these are called schemata, by using '$*$' for 'don't care' in one or more positions. For example,

$$a(1)*a(3)**\ldots*$$

represents the schemata of all strings with first element $a(1)$ and third element $a(3)$, all other elements being arbitrary. Thus any particular string of length l is an instance of 2^l schemata. If l is only about 20 this is still over a million schemata. An evaluation of just one string therefore yields information about a large number of schemata.

The next ingredients of Holland's model are the operators by which strings are combined to produce new strings. It is the choice of these operators which produces a search strategy that exploits co-adapted sets of structural components already discovered. The three principal operators used by Holland are crossover, inversion, and mutation.

Crossover

Proceeds in three steps:

(1) Two structures $a(1)\ldots a(l)$ and $b(1)\ldots b(l)$ are selected at random from the current population.

(2) A crossover point x, in the range 1 to $l-1$ is selected, again at random.

(3) Two new structures:

$$a(1)a(2)\ldots a(x)b(x+1)b(x+2)\ldots b(l)$$
$$b(1)b(2)\ldots b(x)a(x+1)a(x+2)\ldots a(l)$$

are formed.

In modifying the pool of schemata, crossing over continually

introduces new schemata for trial whilst testing extant schemata in new contexts. It can be shown that each crossing over affects a great number of schemata.

Inversion
For some randomly selected positions $x < y$ in the string we perform the transformation:

$$a(1)a(2)\ldots a(l) \rightarrow a(1)\ldots a(x)a(y-1)a(y-2)\ldots a(x+1)a(y)\ldots a(l).$$

Inversion increases the effectiveness of crossover by promoting close linkage between successful alleles (instantiations of string components). Linkage occurs when co-adapted alleles are close together in the genotype, thus reducing the probability that the group will be separated by crossover. This requires an order free string representation and a mechanism for making strings homologous before crossover (see [9] p. 109). The effects of inversion are only apparent over a relatively long time scale, ie a large number of generations. For the purposes of the present discussion inversion may be ignored; our inversion was merely a rather brutal mutation.

Mutation
Each structure $a(1)a(2)\ldots a(l)$ in the population is operated upon as follows. Position x is modified, with probability p independent of the other positions, so that the string is replaced by

$$a(1)a(2)\ldots a(x-1)za(x+1)\ldots a(l),$$

where z is drawn at random from the possible values. If p is the probability of mutation at a single position, then the probability of h mutations in a given string is determined by a Poisson distribution with parameter p. Mutation is a 'background' operator, assuring that the crossover operator has a full range of alleles so that the adaptive plan is not trapped on local optima.

The basic paradigm of a program of this type is as follows:

(1) Randomly generate a population of M strings

$$S(0) = \{s(1,0),\ldots,s(M,0)\}.$$

(2) For each $s(i,t)$ in $S(t)$, compute and save its measure of utility $v(s(i,t))$.

(3) For each $s(i,t)$ in $S(t)$ compute the selection probability defined by

$$p(i,t) = v(s(i,t))/(\text{sum over } i \text{ of } v(s(i,t))).$$

(4) Select a string $s(j,t)$ in $S(t)$ according to the selection probabilities:

- apply crossover with probability P_c to $s(j,t)$ and $s(j',t)$, where $s(j',t)$ is

again selected from $S(t)$ according to the selection probabilities; select one of the two resultants (equally likely) and designate it $s(k, t)$;

- apply simple inversion with probability P_i to $s(k, t)$. Designate the result $s(k, t)$;
- with probability P_m (small) apply mutation to each element of $s(k, t)$. Designate the result $s(k, t)$.

(5) randomly select a string in $S(t)$, where each string is equally likely to be selected (probability $1/M$), and replace the selected string by $s(k, t)$.

(6) Compute $v(s(k, t))$ and replace the corresponding element in the saved array of values of v.

(5) Goto 3.

The main advantages of this adaptive strategy are:

(a) It concentrates strings increasingly towards schemata that contain structures of above average utility.

(b) Since it works over a knowledge base (i.e. the population of structures) that is distributed over the search space, it is all but immune to getting trapped on local optima.

4.3.2 Optimizing the WISARD mapping

Cavicchio[17] first suggested that genetic algorithms might be used for the selection of suitable detector sets for pattern recognizers. However, Holland's theoretical work[9] was based on representations of solutions as strings, where each component of the string has a precise, position-dependent meaning. WISARD mappings as solution strings (in common with many other pattern detectors) substantially lack these semantics of position. Brindle discusses the problem of set representation for the application of genetic algorithms.[18]

The patterns used in training and testing the WISARD simulation consisted of 100 ms of speech data arranged on the retina as 20 columns, each representing successive 5-ms segments, by 16 rows corresponding to a 16-channel filter bank. The imaged data therefore represented a 100-ms sample of speech in the frequency domain.

A WISARD model with $N = 8$ was used. Strings representing mappings therefore consist of 40×8-tuple $= 320$ elements, integers in the range 1–320, see Fig. 10. Unlike many WISARD experiments the random maps used were not 1–1, ie were not necessarily permutations of the integers $1, 2, \ldots, 320$.

To provide a population 50 random strings were generated at the start of an experimental run. The mapping defined by each string was used to train and then test WISARD. The results of testing provide the necessary information from which a measure of fitness can be calculated for each string. By far the most computation time is spent on training and testing in order to calculate the fitness.

Each new string generated by the algorithm therefore requires a complete train and test sequence, typically 40 or 50 training examples per class and around 25 testing samples. It was decided to set the maximum number of iterations to 2000. Even so, each experimental run took around one week on a SUN workstation. With present levels of readily available technology, processing speed and memory, limit the scope of such experiments considerably.

In this context it is instructive to reflect upon the size of the search space. There are 320^{320} possible strings, ie around 10^{801}. Although there are a number of equivalence relations between strings, eg it does not matter in which order the different 8-tuples are placed in the string, or in which order the individual elements of the 8-tuples are placed (these two together effect a reduction by at most a factor of 10^{52}), these do not substantially affect this figure. Generously, assuming a computer capable of testing 10^{10} strings per second it would take approximately 10^{732} years, a time vastly exceeding the estimated age of the universe (a generally accepted upper bound for which is 1.2 times 10^{11} years) to search the entire space exhaustively.

4.3.3 Experimental procedure

The sample speech was collected in the time domain as word pairs, in an attempt to produce some co-articulation effects, and passed through a 16-channel filter bank to produce frequency domain data which was also saved. The frequency information was in 5-ms steps and stored on a VAX 11–750 as a file of unsigned bytes.

For both the training and test phases it was necessary to create a parallel file containing an indication at each step as to which class the 5 ms sample corresponded (A, E, I, O, U or 'no class'). This second file was hand crafted and identification was accomplished by traversing the time domain data in small steps and inspecting a defined area of the file both visually and acoustically.

Variation in vowel length was typically from 45–250 ms, but most vowels tended to be around the 100-ms mark—hence the choice of 100 ms for the retina. No attempt was made to time normalize the data. Other experiments, mentioned earlier, using time normalized training and test data suggest that time variation is the principal limitation on accuracy for this type of task, and we were interested to see to what extent the techniques discussed here could accommodate to this problem. Yet another reason for this decision was that if the initial recognition performance were too good it would not be possible to observe the improvements, if any, effected by the genetic algorithm.

Below is a summary of the vowel sounds used in the word set (in British 'English' as opposed to N. American 'English').

Vowel	Number in training	Number in testing
A as in fAte	47	26
E as in mEt	46	24
I as in bIt	53	27
O as in gOat	52	24
U as in dUe	40	22

Total storage of the speech data in the frequency domain occupies about 10 Mbytes.

As a final stage in pre-processing the data, the 16 unsigned bytes (0–255) at each 5-ms step, each byte representing the intensity of activity in a frequency channel, were converted to a single thresholded bit. Obviously in doing this, much of the original information is lost, but speech recognition involves selective data reduction on a massive scale and generally we have found that it is the presence or absence of activity, rather than the intensity, which is significant in a particular frequency channel.

To determine appropriate thresholds we calculated arithmetic means for each class at each of the 16 frequencies over all the samples in the training data. The processing of sample data in the experiment proceeded by replacing a particular unsigned byte by 0 if its value was lower than the corresponding threshold and by 1 if the value was greater than the threshold.

Given the size of the retina, 320 pixels, the 8-tuple system requires one set of 40 256-bit (eight address lines) RAMs for each of the five classes; a mere 1280 bytes per recognizer or 6400 bytes altogether. One could improve the performance of such a system by increasing the retinal coverage, which is not in any event 1–1, and using more RAM. However, in the context of the present experiment this would increase the length of the string required to describe the mapping, exponentially increase the size of the search space and significantly reduce the rate of convergence. Since 50 WISARD systems are used (recall the population consists of 50 strings) in the experiment the total memory required for RAM is 312.5 Kbytes.

Twenty time slices of data from the frequency file were written onto the retina and moved up one column at each stage. Training and testing were done when the first column of the retina was positioned at the start of a vowel as indicated by the parallel file.

The genetic algorithm, as sketched above, was applied to a population of 50 strings with operator probabilities of

$$P_c = 0.06, P_i = 0.06, P_m = 0.005.$$

Two different measures of utility were tried:

(i) The first measure was chosen to select for 'orthogonality' of discriminator responses, ie two conflicting requirements on each discriminator, a high score on samples from the correct class and a low score on samples from other classes, were combined into one global measure of utility across all discriminators. We do this as follows.

For a given mapping **M** if $r(i, j) \geqslant 0$ is the score of discriminator i, on a sample from class j, $1 \leqslant i, j \leqslant 5$, normalized to the range [0, 1], then

$$\cos A = r(j,j)/\sqrt{(r(1,j)^2 + r(2,j)^2 + \ldots + r(5,j)^2)}$$

is a measure of how far from ideal ($\cos A = 1$) is the response of the whole system to the sample from class j; it measures the angle A between the vector of discriminator responses and the ideal vector. Let the average of $\cos A$ for a single class j over the test samples be $C(j)$. Then we define

$$U(\mathbf{M}) = (C(1) + C(2) + \ldots + C(5))/5.$$

This is taken as the measure of utility of the mapping **M**; **M** would be an ideal mapping if $U(\mathbf{M}) = 1$. We note that any positive monotonic transformation $Q(U)$ of this function will also correctly measure the utility of **M**.

(ii) The second measure of utility was less subtle. In this case the strings were bred simply to maximize the average accuracy of discriminator responses across all classes. Thus for any particular mapping **M** the response to the test samples in class j can be taken as

$$N_c/T_j,$$

where N_c is the number of correct classifications and T_j is the total number of test samples in class j. This figure averaged over all classes j gives a measure of utility $V(\mathbf{M})$ for the mapping **M**. Once more, **M** is ideal if $V(\mathbf{M}) = 1$.

We found the adaptive search procedure worked more efficiently if a positive monotonic transformation was carried out on the utility measure in order to increase the probability of a superior string contributing to the next generation and decrease the probability of an inferior string contributing to the next generation. To do this we based the probability of selection on utility raised to the power 16.

4.3.4 Summary of results

Orthogonality

Fig. 11 shows the measure of orthogonality across 2000 generations (iterations). Breeding for orthogonality is a more difficult task than merely breeding for accuracy. However it can be argued that this might be more important in a real system which is scanning continuously rather

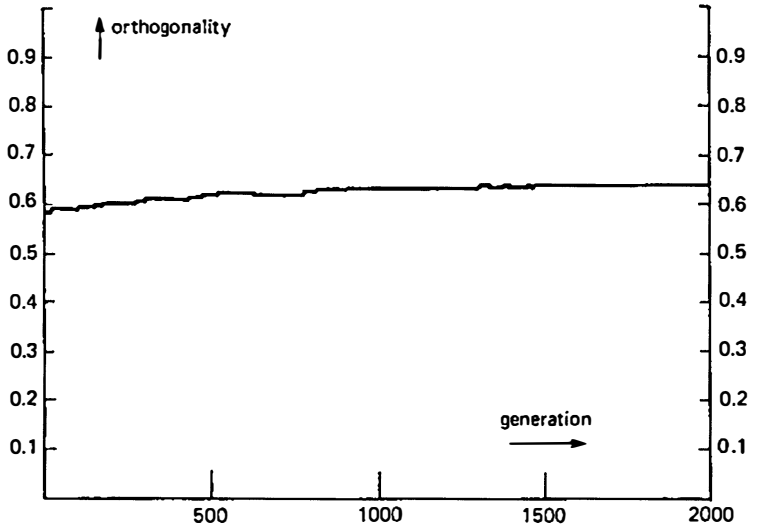


Figure 11 *Breeding for orthogonality.*

than just looking at test vowels, since it may help to suppress spurious discriminator response to features that do not discriminate vowels, ie increase selectively. In fact we found that, although there was measurable improvement in orthogonality, from 0.583 51 to 0.638 75, this did not correlate well with accuracy (which decreased).

Accuracy

In Fig. 12 the average percentage accuracy across all classes, for the best string, is plotted against the current generation. The improvement is significant but not startling—an improvement from 58.7–65.3% across 2000 generations. Broken down across classes the results were as shown in Table 14. Examination of the best string of the search showed that the

Table 14. *Effect of Holland's genetic algorithm*

Vowel	Percentage recognition accuracy	
	Initial best string	Final best string
A	61.5	65.4
E	37.5	45.8
I	59.3	63.0
O	62.5	75.0
U	72.7	77.3
Average	58.7	65.3

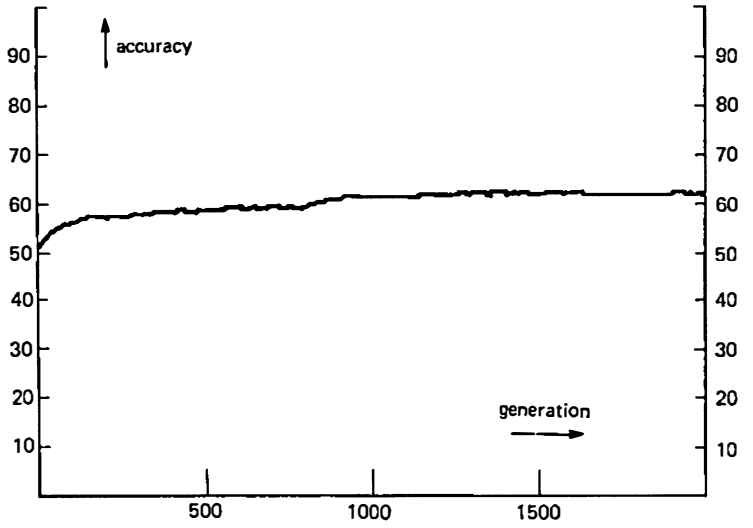


Figure 12 *Breeding for accuracy.*

pixels selected for the 8-tuples were evenly scattered over the retina. No specific frequencies or time slots seemed to be particularly favoured.

The overall effect across the entire population, rather than individual best strings, can be observed in the histograms of Fig. 13. The top row represents the initial performance, on each vowel, of the entire

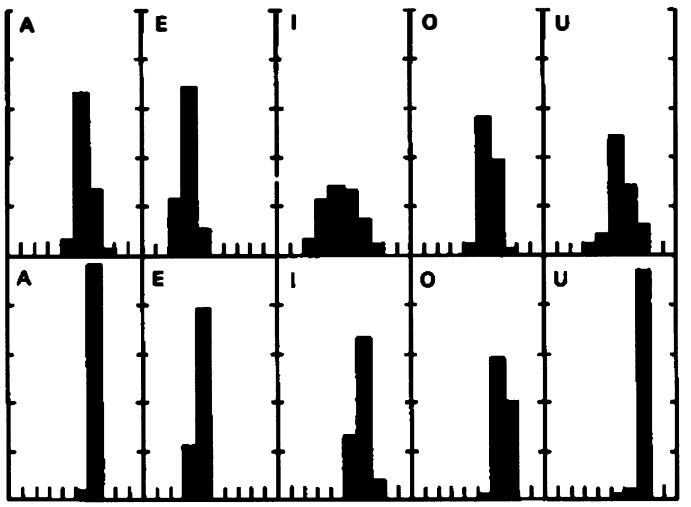


Figure 13 *The overall effect on the entire population.*

population. Horizontal divisions are percentage accuracies in the ranges 0–10, 10–20, ..., 90–100. The vertical scale represents frequency (total—50 strings in each case). Note there was one initial string giving a recognition in the range 70–80% for *A* (presumably a worse performance on other vowels) but in the final population no such string existed. This results from breeding for *average* accuracy across recognition classes. How significant is this improvement in accuracy compared with that which might be expected from a random search through 2050 strings?

Assuming that the distribution of average accuracy across the population of random strings is normal (an assumption reasonably in accordance with the observed facts) we can approximately gauge the efficiency of the search as follows. We first calculate the mean, M, and standard deviation, S, for the initial population of average accuracies and then compute

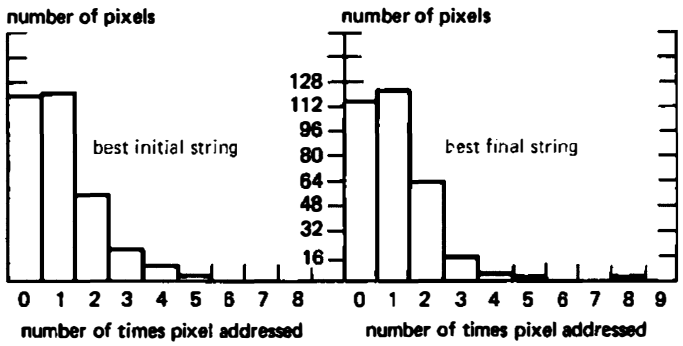
$$(B-M)/S,$$

where B is the average accuracy for the best final string. The figure obtained was 3.92 standard deviations.

The probability that at least one string is as good, or better, than the best string found after a random search through 2050 strings is

$$1-(\text{Area to left of } 3.92 \text{ on normal distribution})^{2050},$$

ie approximately 0.1. By this measure the genetic algorithm was about ten times more efficient than an exhaustive search.



Addressed	0	1	2	3	4	5	6	7	8	9
Initial	119	120	53	19	8	1	0	0	0	0
Final	115	122	62	15	4	1	0	0	1	0
Poisson	118	118	59	20	5	1	0	0	0	0

Figure 14 *Initial and final distributions of pixel addressing.*

We were interested to see if there was a significant change in the distribution of frequency with which individual retinal pixels are sampled. Given an initially random selection of pixels one might expect the frequency with which an individual pixel is addressed to be Poisson with mean 1. In Fig. 14 the initial and final distributions for the best strings are given and compared with the expected Poisson distribution. From this it can be seen that no significant shift occurred after 2000 generations.

If there were a performance advantage to be gained from a 1–1 mapping one would expect the number of pixels addressed zero times to decrease with the number of generations, whereas Fig. 14 shows that this effect did not occur in any significant way.

4.4 Conclusions

If we compare the results of the multiple discriminator per vowel experiment with the genetic algorithm experiment, some interesting points emerge.

Vowel	A	E	I	O	U
MDV	61.5	54.2	59.3	54.2	81.8
IGA	61.5	37.5	59.3	62.5	72.7
FGA	65.4	45.8	63.0	75.0	77.3

MDV = multiple discriminators per vowel.
IGA = initial performance in genetic algorithm experiment.
FGA = final performance in genetic algorithm experiment.

The 'most difficult' vowel E and the 'easiest' vowel U have plainly benefited in the first vowel experiment from having discriminators trained over a variation of timescales. On the other hand, the genetic algorithm was able to tailor a *single* mapping to compensate to a significant degree for variations in rate of articulation of these vowels.

However, while some improvement in average accuracy of recognition across classes can be effected using genetic algorithms, the bunching of the population in the histograms of Fig. 13 demonstrates that for each vowel there is a definite limit beyond which no improvement can be expected using a single mapping for all classes. Indeed, from Fig. 13, we can roughly estimate the upper bound for average accuracy for this system as lying in the range 72–82%.

The evidence seems to suggest that the principal reason for this upper limit in performance is the time variation in the samples used. By using a *separate mapping for each class* the results using genetic algorithms may well improve to the point where it would not be necessary to have

multiple discriminators per class. However, once the extra overhead of a separate mapping for each class is accepted, then other, simpler, possibilities exist for optimizing mappings which should be explored first, possibly keeping the genetic algorithm in reserve for fine tuning at the final stage.

Certainly genetic algorithms offer considerable gains in efficiency over exhaustive search in tailoring pattern recognition systems operating on real data. The improved mappings produced by the genetic algorithm showed no tendency to become 1–1. We found this an interesting observation but hesitate to draw a general conclusion.

Acknowledgement

The authors gratefully acknowledge the support of British Telecom Research Laboratories R 18.3.2, Martlesham, under contract number MSP3231A/25 Control 318028, for major parts of the work described in this chapter. We are also indebted to IEE Proceedings for kind permission to reproduce much of [2].

References

1. Bledsoe, W. W. & Browning, I. Pattern recognition and reading by machine, *Proc. Eastern Joint Computer Conf.* (Boston, Mass: 1959).
2. Binstead, M. J. & Jones, A. J. A design technique for dynamically evolving *N*-tuple nets, *IEE Proceedings*, Vol. 134, Part E, No. 6, pp. 265–269 (November, 1987).
3. Minsky, M. & Papert, S. *Perceptrons—An Introduction to Computational Geometry* (Cambridge Mass: MIT Press, 1969).
4. Aleksander, I. & Stonham, T. J. A guide to pattern recognition using random-access memories, *IEEE Journal Computers and Digital Techniques* **2**(1), 29–40 (1979).
5. Kohonen, T. *Self-Organisation and Associative Memory* (Berlin Springer Verlag, 1984).
6. Barto, A. G. & Sutton, R. S. *Goal Seeking Components for Adaptive Intelligence: An Initial Assessment* (University for Massachusetts, Amhurst: Technical Report no. AFWAL-TR-81-1070, 1981).
7. Fish, A. N. The conformon: a synaptic model of learning (Ph.D. thesis, University of Manchester, Department of Psychology, 1981).
8. Tattershall, G. D. & Johnson, R. D. Speech recognition based on *N*-tuple sampling, *Proc. Spring. Conf. Inst. Acoustics Swansea*, Vol. 9, No. 2 (April, 1984).
9. Holland, J. H. *Adaptation in Natural and Artificial Systems* (University of Michigan Press, 1975).
10. Holmes, J. N. The JSRU channel vocoder, *IEE Proceedings*, Vol. 127, Part F, No. 1, pp. 53–60 (February, 1980).
11. Bellman, R. E. *Dynamic Programming* (Princeton University Press, 1957).
12. Vintsyuk, T. K. Element by element recognition of continuous speech composed of the words of a given vocabulary, *Kibernetica* **2**, 133–143 (1971).
13. Sakoe, H. & Chiba, S. A dynamic programming approach to continuous speech recognition, *Proc. of Int. Cong. of Acoust., Budapest, Hungary*, pp. 200–213 (1971).
14. McClelland, J. L. & Rumelhart, D. E. (eds) *Parallel Distributed Processing*, Vol. 2 (Cambridge Mass: MIT Press, 1986).

15. Leszek, Kot. A syntax-controlled segmentation of speech on the basis of dynamic spectra, *Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 2015–2017 (1982).
16. Jassem, W. Speech recognition work in Poland, Chapter 23 of *Trends in Speech Recognition* (Prentice-Hall, pp. 499–511, 1980).
17. Cavicchio, D. J. Adaptive search using simulated evolution (PhD thesis, University of Michigan, 1970).
18. Brindle, A. Genetic algorithms for function optimization (PhD thesis, University of Alberta, 1980).