Experiments in Isolated Digit Recognition with a Cochlear Model

Eric P. Loeb and Richard F. Lyon
Schlumberger Palo Alto Research
3340 Hillview Avenue
Palo Alto, CA 94304

Abstract

We have conducted speaker-independent isolated digit recognition experiments using vector quantized cochleagrams. Without the use of time order information, we were able to achieve a recognition rate of 97.24%. With a modified Viterbi algorithm we achieved a rate of 98.38%. Since we achieved a 98.05% recognition rate with a scheme that did static pattern matching on the first and second time-halves of our utterances, we must call into question the effectiveness with which the Viterbi algorithm uses time order information. Our results also lead us to conclude that future progress may depend on our ability to construct more sophisticated vector quantizers.

1 Introduction

We have conducted a number of isolated digit recognition experiments in an effort to evaluate the potential of an auditory model front end. The experiments emphasize non-parametric approaches and techniques that use little or no time order information. We wished to set high performance standards for future experiments while estimating the relative importance of the various sources of information in the data.

Although the front end for our experiments is a cochlear model [Lyon '82], there is nothing explicitly neural about our techniques. They could be applied to any other vector quantized representation. Many of the experiments are interesting as techniques for the use of non-parametric statistics in spite of the shortage of training data. Since every experiment uses the same training and testing data sets, all the results are directly comparable.

2 General Methods

The first repetitions of the isolated digits in the training subset of the TI Connected Digit Database (sampling rate 20kHz) were analyzed by our cochlear model. The model produces a discrete-time 92-channel spectrum, which is down-sampled to 1 kHz and quantized by a standard Euclidean quantizer with 1024 codewords. The quantizer codebook was trained on the entire corpus using the standard K-means algorithm.

In all the experiments to be described, half of the speakers were used for training and the other half were used for testing. Thus the recognition results in this paper are ostensibly speaker-independent. We can not claim total speaker-independence because we used both sets of speakers to build our vector quantizer. Since this caused two of the codewords never to occur in the training set, the net result may actually be poorer performance than we would otherwise expect. At any rate, all experiments used the same training and testing sets, so all results are directly comparable.

Recognition Method	Grouping			Time Order
	Original	0.95	0.80	
Basic Method	94.97%	95.45%	95.29%	No
Necessity	94.16%	95.62%	96.10%	No
Conditionals	—	96.27%	94.81%	No
Ranges	94.81%	97.24%	95.94%	No
Neural Net Model	—	93.18%	—	No
Time Splitting	96.10%	95.94%	96.27%	Some
...with Necessity	94.81%	96.43%	97.56%	Some
...with Ranges	95.78%	98.05%	97.56%	Some
Viterbi Algorithm	98.38%	94.15%	—	Yes

Table 1: Recognition percentage for 616 test utterances. Grouping numbers are the *coverage-proportion* values from section 5.1

3 Definitions

codeword an integer in [0, B] where B $\leq$ 1023. Each codeword corresponds to some subset of $\Re^{92}$, and the set of codewords corresponds to a partition of $\Re^{92}$.

utterance the sequence of codewords derived from the cochleagram of one of the speakers saying one of the vocabulary words.

utterance histogram a vector $\vec{H}(A)$, where $\vec{H}_{cw}(A)$ is the number of occurrences of codeword cw in utterance A.

guess the index to a vocabulary word. A guess is the result of some recognition method operating on a test utterance.

guess vector a <vocabulary-size>-dimensional vector of word log probabilities. **If a recognition method generates a guess vector, then it will always output the index of the most probable word as its guess.**

4 The Basic Method

It is useful to consider this simple non-time-order method in several different ways. A matrix of conditional log probabilities of observations (codewords) given the word hypothesis is first generated. Probabilites are estimated from a count of the number of occurrences of each codeword within all training utterances of each vocabulary word. Then, for recognition, each word in the vocabulary is scored by adding the log likelihoods for all time samples in the unknown utterance; since these scores do not depend on the order of occurrence of the samples, they are most easily computed by multiplying the log probability matrix by the utterance histogram.

Now each codeword, cw, indexes a vector $\vec{V}(cw)$ in our matrix, where

$$\vec{V}(cw)_i = -\log \Pr[\text{codeword } cw \mid \text{word } i]$$

With this in mind, it is clear that we can form the same guess vector by summing the scalar products of each $\vec{V}(cw_k)$ times the projection of the test utterance histogram onto the unit vector corresponding to cw_k. We will return to this in sections 5.2, 5.4 and 7.

Another way to form the same guess vector is to accumulate the $\vec{V}(cw)$'s indexed by each codeword found in sequence in the test utterance.

Finally, consider an equivalent neural network. Each codeword corresponds to an input neuron, and each vocabulary word corresponds to an output neuron. An input neuron fires once each time its codeword occurs in the test utterance. Input neuron *cw* is connected to output neuron *i* by a linear excitatory synapse of weight $\vec{V}(cw)_i$ Output neurons sum their inputs, and the number of the cell with the highest value is our guess.

This simple program gives 94.97% correct recognition on the testing set (see table 1). This implies that the codewords (and thus the underlying cochleagrams) are doing a good job of acoustically separating our vocabulary words.

It is interesting to note that an earlier version of the codebook, in which the K-means algorithm had not iterated to convergence, gave 94.16% recognition. This indicates that the method of codebook vector production is an important component of a quantizer-based system.

5 Simple Variations on the Basic Method

5.1 Codeword Grouping

Let us suppose there are several codewords covering the spectra produced by /s/ sounds. Then the majority of the observations of these codewords will occur in the vocabulary words containing /s/ sounds. In the task at hand these words are *six* and *seven*. So, if we assign one new number to every codeword, cw, such that most of the observations of cw occur in the words *six* and *seven*, then this new number should be a good indicator of the /s/ sound.

To implement this idea we will need a parameter *coverage-proportion*. We will map each codeword, *cw*, to the set of vocabulary words that account for at least *coverage-proportion* of the observations of *cw*. Next we map these sets to integers (i.e., we number them). The composition of these two mappings is a many to one map from the original codewords to some new codewords. We then use the new codewords in the basic method.

Note that codeword grouping may make signal reconstruction impossible.

The results are on the "Basic" row of table 1. The *coverage-proportion* = 0.95 grouping reduces the number of codewords from the original 1024 to 405. The 0.80 grouping has 313 codewords. It maps 71 of the original codewords to the set {6 , 7}, 39 originals to the set {3 , zero}, and 22 to {1 , 9}. Although many of the sets do not have such obvious phonetic content, most of the sets that represent large numbers of original codewords do.

Thus we may have found a way to generate phonetically meaningful labels without imposing our pre-conceptions upon the data. In the future, we hope to extend this method to the grouping of sequences of codewords.

5.2 Non-Occurring Codewords (or Necessity)

Our basic method will often guess "zero" when the input is an "oh". The reason for this is that the method is one of sufficiency, in that we have no way of necessitating a /z/ sound before guessing "zero". Thus we wish to make use of the codewords that do not occur.

To do this we will start with the basic method. Then, for each codeword, cw, that does not occur in the test utterance we will add an inverse of $\vec{V}(cw)$ (see basic method) to our guess vector. We inverted $\vec{V}(cw)$ by subtracting each element from the vector maximum.

Notice that this method adds a non-linearity to our system. Our probability estimates given codeword *cw* are no longer 0 when the number of occurrences of *cw* equals 0.

If we examine the differences between the "Basic" and "Necessity" results in table 1, then it appears that this method becomes more successful as the number of codewords decreases. If this were the case, however, we would expect a large improvement in the 0.50 grouping, which has only 156 codewords. The recognition rates with this grouping were 90.26% with the basic method and 91.56% with the current method. So the utility of this method seems to depend on the extent to which our codewords correspond to phonetic units.

It is provocative that this improved method corresponds to a more realistic neural network model. In real sensory processing systems, when one finds a neuron that responds to a given event, one often finds another neuron that is inhibited by that event. A network model for the necessity method will have two neurons for each codeword. One neuron fires once for each of its codeword's occurrences, and the other fires when the codeword does not occur.

5.3 Second Order Conditionals

It is incorrect to assume the codewords are independent, but to do otherwise we must consider higher order conditionals such as the probability of word X given codeword A, codeword B, and no codeword C. The number of conditionals of this form is, however, prohibitively huge. So we will restrict ourselves to pairs of occurring and non-occurring codewords.

To do this we essentially use the basic method as many times as there are codewords. (see section 4) We make two matrixes of log probabilities for each codeword. One is created by and for utterances in which the codeword occurs, and the other matrix is for utterances in which the codeword does not occur. For each codeword, we multiply the appropriate matrix by the histogram of the test utterance to get a guess vector. These intermediate guess vectors are then summed to get the final guess vector.

We tried this method with several different groupings. The 0.80 grouping was the only one that did not show improvement. (see table 1) This method uses far too much memory to be useful, but the fact that such a simplistic scheme for utilizing the inter-codeword correlations gave improved performance does suggest that this is a rich source of further improvements.

It is interesting that the use of a single large matrix containing one log probability vector (i.e., one row) for every possible pair of codewords (both occurring and non-occurring) did not work. Performance with the 0.95 grouping was degraded from 95.45% (basic method) to 92.86%. The loss in performance is probably due to our inability to make a meaningful count of the number of occurrences of a pair of codewords.

5.4 Several Ranges for Each Codeword

Recall that our neural network model has cells that fire when their codewords do not occur, and others that fire in direct proportion to the number of occurrences when they do occur. In a real neural system, we would expect one neuron to fire when the codeword does not occur, one to be sensitive to a small number of occurrences, another to fire in proportion to a larger range of occurrences, and another that only fires during large inputs. In general, such neural pools seem to divide the numbers they encode into approximate log ranges [Brooks '86, chapters 3 and 4].

To implement this, we used the basic method with 6 matrices. If we let n_k be the number of occurrences of codeword k in utterance A, then utterance A will contribute to or use the $\lfloor \log(1+2n_k) \rfloor$th matrix for codeword k.

This method makes our system more non-linear, since (see section 4) our probability vectors $\vec{V}(cw)$ are now functions of the number of occurrences of cw as well. So this improvement may be due to our adding more detail to the conditional probabilities. Alternatively, it may be due to the fact that we are now taking into account the average number of occurrences of a codeword among those utterances in which it occurs. This statistic represents durational information. We used it in the previous methods only indirectly.

6 Back Propagation Networks

We wished to compare an actual neural network model to our other systems [Plaut et. al. '86]. This model, called a back propagation network, is made up of layers. Each layer has an input vector, V_I, an output vector, V_O, and a matrix mapping V_I to V_O. Each layer functions by mapping V_I to V_O with its matrix, and then running V_O through a component-wise non-linearity, $f(x) = 1/(1+\exp(-x))$. The resulting V_O is now the input to the next layer. It is easy to train such a network to perform any vector transformation.

We began our experiments with a simple 1-layer network that mapped normalized histograms of utterances to <vocabulary-size>-dimensional vectors. In order to reduce computation, we used the 0.95 grouping. As in the basic method, the argument of the maximum value in the output vector was our guess. For most of the experiments each input element was the ratio of the number of occurrences of its codeword to the total number of occurrences of all codewords. In another set of experiments now under way we are using 6 input elements per codeword with the encoding of section 5.4.

The simple 1-layer network converged to 92.86% correct on the testing set, with over 99% correct on the training set. When we added a second 11x11 layer which took the simple network's output as its input, the new network could learn which words were easily confused. This caused the recognition rate to go to 100% on the training set, but down to 92.5% on the testing set. So, this network over-learned the training data, and failed to generalize.

In order to force some generalization, we used the same data on a 2-layer network that had a 6-D vector in the middle. This structure resulted in 1 error on the training set and 93.18% correct on the testing set (see table 1). Since the large network over-learned the training data, and the smaller network did not greatly improve performance, it seems unlikely that back propagation networks can be used to great advantage on this particular problem.

7 Vector Quantization Methods

Let us suppose we have a speech recognizer box. Its input is a speech waveform or sequence of observations, which may be thought of as a vector. Its output is a word, which may be thought of as a scalar. Thus our speech recognizer is, in fact, a vector quantizer. Can it be implemented directly as one?

To cut down the pattern space some, we use binary histograms (i.e., each codeword either occurs or does not occur in the test utterance) with Euclidean distance, and the standard K-means algorithm to construct a codebook. A test utterance then maps to its closest codebook vector, which in turn tells us which vocabulary word to guess (the one that most frequently mapped to that codebook vector in training).

In a second experiment codebook vector k was set to be the centroid of all the training vectors for vocabulary word k. In another experiment we formed 32 ortho-normal vectors from the 32 codebook vectors of the first experiment. We then used the basic method by finding the projection of the test utterance on each of these vectors and summing the product of these numbers and the appropriate log probability vectors.

In the first experiment, a codebook of size 16 gave recognition = 66.6%. When size = 32, recognition = 76.5%, and when size = 128, recognition = 80.5%.

In the second experiment, with one codeword per vocabulary word, we got 92.69% recognition. This gives a rough idea of the efficacy of the K-means algorithm in approximating the "correct" decision boundaries.

The third experiment gave recognition = 70.13%. Since the codebook vectors we ortho-normalized for this experiment were the same 32 vectors used in the 32 vector part of the first experiment, it is clear that this method was of no help whatsoever.

8 Time Order Methods

8.1 Time Splitting

As a simple extension of the methods we have tried so far, we will use the basic method on the first and second time-halves of the utterances. Thus each test utterance will produce two guess vectors : one for its beginning and one for its end. The final guess vector will be the sum of these. In a second experiment, we use the necessity look-up method (section 5.2) on both time-halves. In a third experiment, we use the Range method (section 5.4) on both time-halves.

As in section 5.3, the fact that such a simple method could provide so much of an improvement (see table 1), confirms that the time order information will be extremely helpful when used properly. The results of the combined time splitting and codeword ranges methods are respectable, but they depend far too heavily on the grouping parameter to be considered useful.

8.2 Viterbi Algorithm

The Viterbi algorithm is well known in speech recognition. We have applied it using simple finite-state word models similar to those used by Bush and Kopec [Bush '85].

The cost metric used by the Viterbi algorithm in finding a best model-based segmentation is $-\log \Pr[\text{codeword} \mid \text{state}]$, as in our basic method. The state tables were initially trained using segmentations found by Bush and Kopec's LPC-based recognizer; they have been retrained and modified to improve performance. In comparing the fits of the various word models, we used measures other than total cost (probability), as described elsewhere [Lyon '87].

The scores reported in table 1 are the best of several variations. Other variations on the scoring function, for example using total Viterbi cost or omitting durational probabilities, resulted in up to twice as many errors. We were able to reduce the error rate from 1.62% to 1.06% using the same codebook but twice as many training repetitions, but we have not yet tested how this would effect the performance of our other algorithms.

It is interesting that the finite-state models give a performance that is at best only slightly better than the techniques that use little or no time sequence information. Better techniques for handling timing are still needed.

9 Conclusions

It is clear that our cochlear model provides an adequate, if not superior, spectral representation. The unexpectedly good performance of the simple methods implies that the cochleagrams are effectively separating phonetic units. Our back end processing is not very sophisticated by comparison.

It is important to note that the experiments presented here are only a fraction of the things we have tried. This lends significance to the non-time-order methods that actually worked. They can be described as an OR operation (section 5.1), a NOT operation (section 5.2), an AND operation (section 5.3), and a kind of subset operation (section 5.4). If we suppose that repeated performance of these operations will continue to improve recognition rates, then we come to the sensible conclusion that we will get our best performance if we can estimate the continuous density that gives the probability of each word at each point in the space in which the utterance histograms reside. Since we always guess the most probable word, this would simply be a partition of that space into word-regions. But this is once again a simple vector quantizer!

This is a problem. The reasoning of section 7 and our experimental results, and common sense all point to that the best system being some kind of vector quantizer. However, the results of our direct experiments were remarkably poor. We may simply have too little training data, or we may never have enough data for the K-means algorithm to work well. We suspect that the crucial difference between a Euclidean quantizer and our basic method is the weighting the basic method gives to each of the codewords. This suggests some experiments, but the net impression one gets is that we will need quantizers that learn the best metric for performing a given quantization. The neural network models appear promising in that regard.

We have not yet tried training separate word models for males and females, which had been found to significantly reduce errors in the previous LPC-based recognizer. Separate training by gender or by dialect could improve any of the methods reported.

The most difficult remaining question is how to handle time order information. The proximity of our Viterbi results to the results that used little or no time order information forces us to conclude either that time order information is not so useful as had been thought, or that the Viterbi algorithm with simple word models does not use it very effectively. We see little difference between the static "codeword A and codeword B" and the dynamic "codeword A and then codeword B." Perhaps a good means of taking into account the co-occurrence of codewords will necessarily lead to a good means of handling timing information. Indeed, does any time-order method not simply amount to a static pattern matching on a trajectory space?

References

[Brooks '86] Vernon B. Brooks. *The Neural Basis of Motor Control*, Oxford University Press, Inc., New York, 1986.

[Bush '85] Marcia Bush and Gary Kopec, "Evaluation of a Network-Based Isolated Digit Recognizer Using the TI Multi-Dialect Database," *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Tampa, March 1985.

[Lyon '82] Richard F. Lyon, "A Computational Model of Filtering, Detection, and Compression in the Cochlea," *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Paris, May 1982.

[Lyon '87] Richard F. Lyon "Speech Recognition in Scale Space," *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Dallas, April 1987 (these proceedings).

[Plaut et. al. '86] David C. Plaut, Steven J. Nowlan, and Geoffrey E. Hinton. "Experiments on Learning by Back Propagation," Technical Report, CMU-CS-86-126, Carnegie-Mellon University, June 1986.

Research supported by DARPA contract #N00039-85-C-0585.