TR－A－0115

Recurrent LVQ for Phoneme Recognition

Erik McDermott and Shigeru Katagiri

1991. 6.12

ATR視聴覚機構研究所

〒619-02 京都府相楽郡精華町乾谷 ☎07749-5-1411

ATR Auditory and Visual Perception Research Laboratories

Inuidani,Sanpeidani, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-7749-5-1411
Facsimile: +81-7749-5-1408
Telex: 5452-516 ATR J

© ㈱ATR視聴覚機構研究所

Recurrent LVQ for Phoneme Recognition

Erik McDermott and Shigeru Katagiri
ATR Auditory & Visual Perception
Research Laboratories
Sanpeidani, Inuidani,
Soraku-gun, Seika-cho,
Kyoto 619-02, Japan

1. Introduction

A key issue in speech recognition is the representation of the temporal structure of the speech signal. In Hidden Markov Models the sequential nature of the speech signal is explicitly represented by matching incoming speech against a connected sequence of states, each of which models speech at a given temporal position. However, explicit modelling of this sort requires that one design the state sequences manually, and decide upon the appropriate number and connectivity of the states. It might be advantageous to learn how to represent temporal structure implicitly. Recurrent "neural" networks are a promising method for achieving this [6,7,8].

In previous work [11] we reported high recognition rates for simple LVQ (Learning Vector Quantization) networks trained to recognize phoneme tokens that are shifted in time. To represent the acoustic context, these networks used a fixed-width window which was shifted over the input. In this method, the fixed-width window was assumed to be sufficient to represent the necessary context. However, the fact that the length of the window is fixed means that phonemes that are either longer or shorter than the window will not be optimally represented.

Here we examine whether recurrent LVQ networks can represent context more efficiently.

2. Recurrent LVQ

LVQ is a prototype-based pattern recognition algorithm. Each category of the task is associated with a number of reference vectors; classification is done by nearest neighbour search among all reference vectors. During the training phase, reference vectors are adapted so as to minimize the number of mis-classification. See [4, 12] for a detailed description of the algorithm, and [3] for a theoretical analysis describing LVQ as a method for gradient descent on a loss function reflecting the mis-classification rate.

Figure 1 shows a recurrent LVQ architecture. The speech part of the input shown here is represented as a matrix of a variable number of time

frames of 16 melscale spectrum channels each. These coefficients are displayed in Figure 1 as black or white squares of varying sizes, size representing magnitude, black for positive values, white for negative values.

The idea of recurrent LVQ is to represent the internal state of the network as it scans speech input, and to feed the state representation back to the network, in the form of an additional input vector. For recurrent networks trained using Back-propagation [6,7,8], the internal state representation is simply the vector of hidden unit activations. In recurrent LVQ, instead of hidden unit activations, a function of the vector of distances between the previous input vector and each reference vector was used to generate the internal state representation. As in Shift-Tolerant LVQ, a window is shifted over the speech input, one frame at a time. This time, however, the window is not the only representation of context: it is supplemented by the recurrently generated state vectors. The reference vectors thus use two sources of information to discriminate phonemes: the speech input at each window position, and the recurrently generated internal state vector for the previous window position[1]. This means each reference vector has two parts: a speech part and a context part. The dimensionality of the reference vectors is the number of spectral coefficients inside the window (16 channels times the window width, in frames) plus the number of reference vectors itself.

The training procedure we adopted was as follows:

1) *Initial Conditions:* The speech part of the reference vectors was initialized as in Shift-Tolerant LVQ, by using the K-means distortion minimizing procedure. Due to the recurrent nature of the context units, it was deemed inappropriate to perform K-means clustering on the context part of the reference vectors. The context part was thus initialized at small random values.

2) *LVQ1 training.* The LVQ1 algorithm [1, 4] was first applied to the recurrent architecture (i.e. to both the context and speech parts of the reference vectors), as a prelude to LVQ3 training. LVQ1 has the effect of spreading the reference vectors over the pattern space, with a density proportional to the joint probability density of the input. This constitutes a good initial configuration for LVQ3 training, which concerns itself specifically with reducing the number of mis-classification. Furthermore, LVQ1 can be applied to the recurrent context units more easily than K-means.

3) *LVQ3 training.* Finally, LVQ3 training [12] was applied to the recurrent architecture. The idea is that the context units will provide useful information about the time course of phoneme utterances, information that will help discriminate phonemes. The distance calculation in recurrent LVQ uses both the recurrent and speech parts of the reference

---

[1] For the very first position of the time window, there is no previous internal state; the context units are thus set at 0.0.

vectors. The resulting distances are then fed back to the next input through the context units.

Training here is performed segmentally; i.e. the task is to discriminate a phoneme given only the information from the context and speech units for a given position of the sliding temporal window. No "back-propagation in time" is performed [8,10] here; rather the method is very similar to that used in [6,7]. This local training method is viewed as an approximation of back-propagation in time.

4) *Recognition:* A phoneme token is recognized using the same method as in Shift-Tolerant LVQ: the distances between input vectors and closest reference vectors for each category are summed over time; the category with the lowest sum is chosen as the recognized category.

Given an N dimensional distance vector (one value for each reference vector), the function used to generate the internal state vector (of equal dimensionality) was of the following form:

$$f(d_n) = 2 \cdot \frac{1/d_n}{\max_{1 \le i \le N} (1/d_i)} \cdot N/S$$

This function is applied to each component of the distance vector. Here d_n is the distance between the input vector and the nth reference vector, N is the total number of reference vectors, and S is the dimensionality of the speech segments obtained at each position of the shifting window. This function has two significant properties: 1) small distances (for reference vectors that are close to the input vector) are mapped to large values; 2) the value range of the resulting N dimensional internal state vector (or context vector) is between 0 and 2N/S. Given the -1 to 1 value range of the S dimensional speech vector [4], the contribution that the context part makes to the distance calculation is thus roughly equal to that of the speech part.

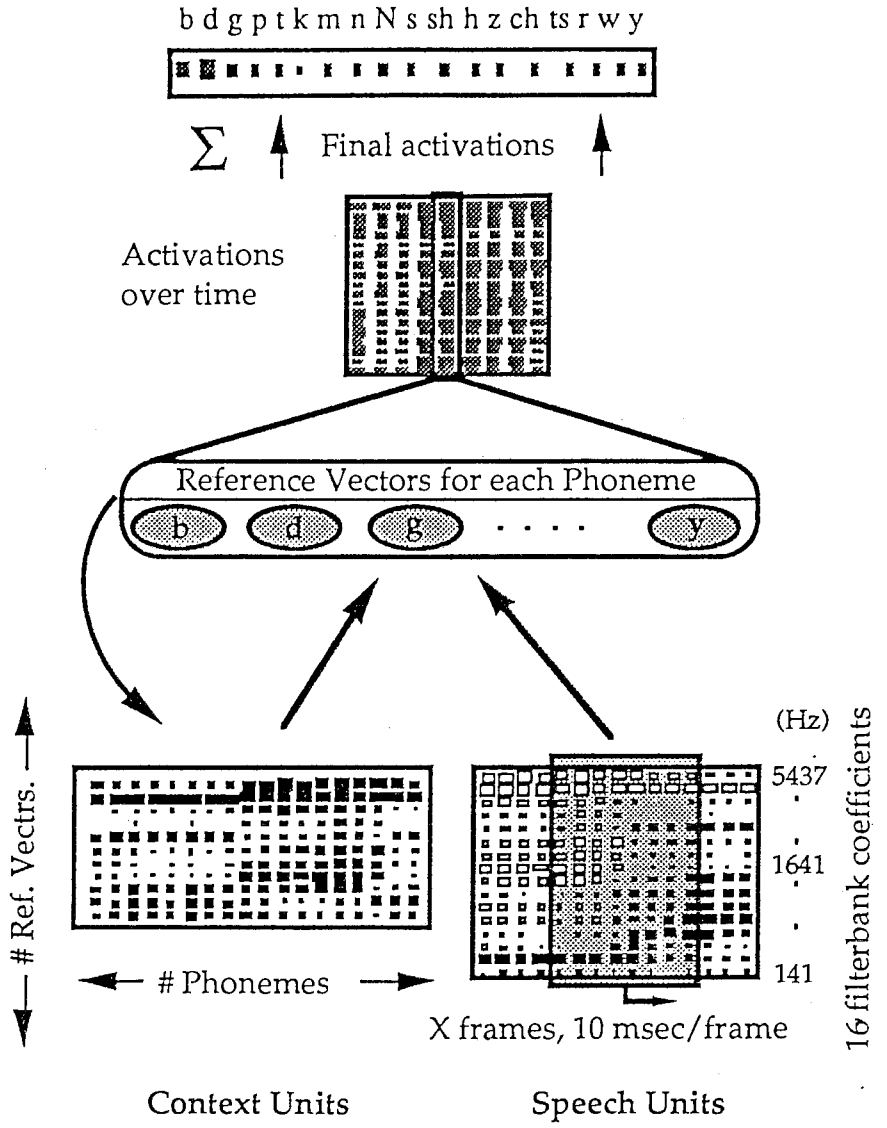


Figure 1. Recurrent LVQ for phoneme recognition

3. Databases and Experiments.

Four databases were examined here:

Speaker-dependent:

A) a database of 5240 Japanese words, spoken in isolation;

B) a database of 880 Japanese sentences, spoken semi-continuously, phrase by phrase;

C) a database of 880 Japanese sentences, spoken continuously;

Half of the first database was used for training; the other half and the other databases were used for testing. All three databases are for a single male speaker.

Speaker-independent:

D) a database of 5240 Japanese words, spoken in isolation by 16 speakers. Data for 15 speakers were used to train the system; data for the remaining speaker were used to test the system. The datasets used for training and testing were of approximately the same size .

The task we examined, using these different datasets, was the recognition of all Japanese consonants. This task was previously used to evaluate the standard, non-recurrent Shift-Tolerant LVQ algorithm [4] [13].

We addressed two questions: 1) can recurrent LVQ achieve better performance than non-recurrent LVQ? 2) can recurrent LVQ using a small time window achieve the same level of performance as non-recurrent LVQ using a larger time window? Positive answers to these questions would suggest that recurrent LVQ has learned to use the context units to represent the time course of a phoneme token.

To answer these questions, we compared the recurrent LVQ architecture with the standard Shift Tolerant LVQ architecture, applied to the 4 databases described above, for two different window sizes: 3 frames and 7 frames. The number of reference vectors was kept fixed, at 15 vectors per category. For the recurrent architecture, LVQ1 was performed for 10 epochs (1 epoch = one full presentation of the training tokens in the task), after which LVQ3 was performed for 15 epochs. For the non-recurrent, standard Shift-Tolerant LVQ architecture, LVQ3 training was performed for 15 epochs. Note that our purpose here is comparison between recurrent and non-recurrent LVQ, and not an attempt to produce the highest absolute recognition levels possible for these algorithms.

Table 1 shows the results obtained after a single learning run (i.e. one performance of the three steps described above) for each combination of architecture and recognition task.

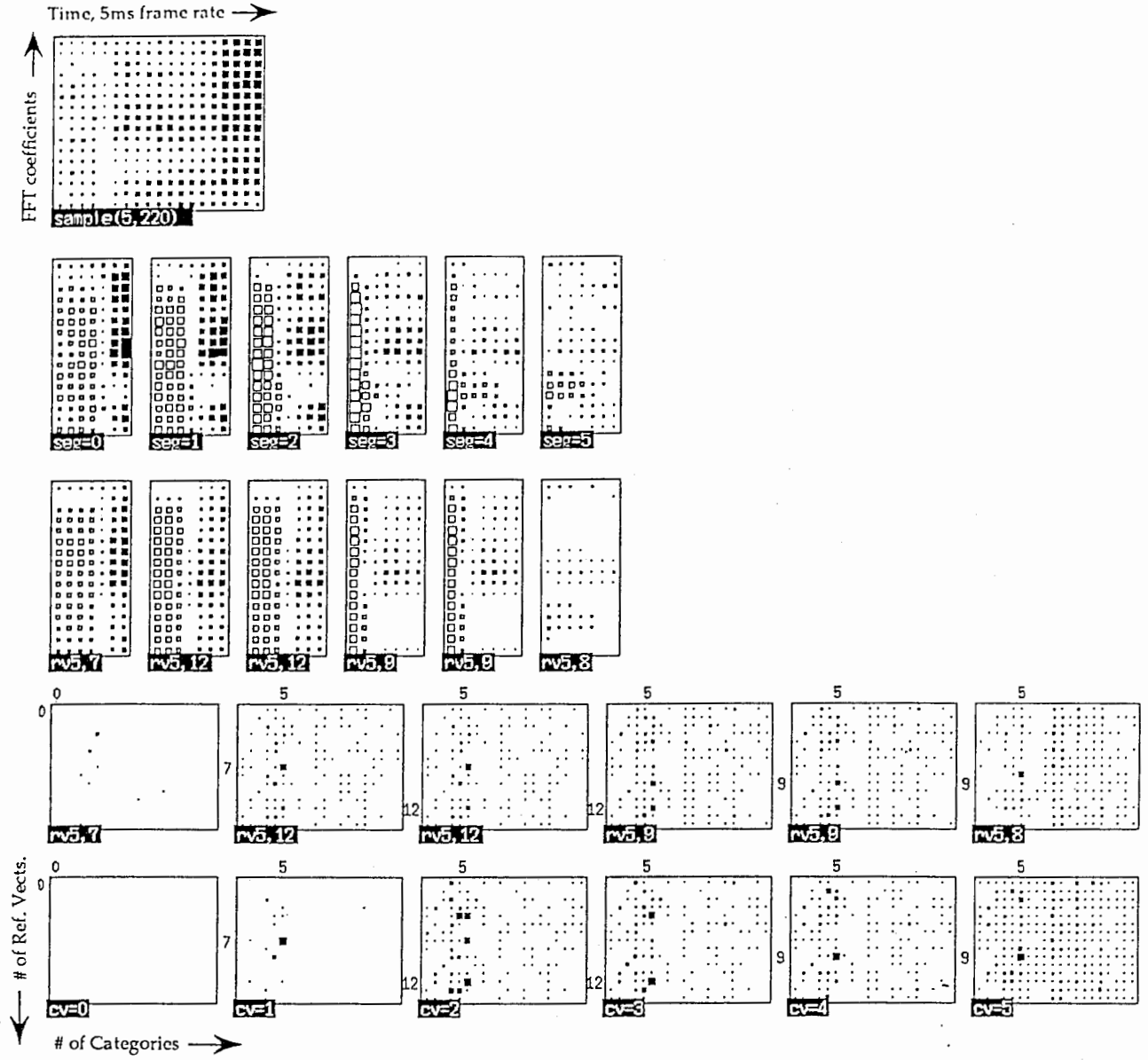
		Recognition Method: Window Size			
		3 frame window		7 frame window	
	Databases	Architecture: recurrent	non-rec.	Architecture: recurrent	non-rec.
Speaker-dependent	A: Isolated Words (train/test)	98.3% / 95.9%	98.0% / 95.5%	99.4% / 97.4%	99.7% / 97.2%
	B: Phrase-by-phrase Utterances (test)	77.9%	79.7%	85.1%	84.2%
	C: Continuous Speech (test)	70.7%	71.3%	74.3%	73.0%
D: **Speaker Independent** Isolated Words (train/test)		82.6% / 76.6%	82.9% / 76.9%	92.9% / 86.0%	92.7% / 86.0%

Table 1: Phoneme recognition results for recurrent and non-recurrent recognition architectures of different window sizes, tested on phonemes taken from 4 different databases.

4. Discussion

The results we obtained suggest that the recurrent architecture did not succeed in learning to represent phonemic context any more efficiently than the non-recurrent architecture does.

For all the tasks examined, the difference in performance between recurrent and non-recurrent architectures is small. The striking aspect of the results in Table 1 is rather the difference between architectures using a 3 frame window and those using a 7 frame window. This confirms what we already know [4], that 7 frames affords a better representation of context than 3 frames does, while providing little support for our hope that recurrent architectures would be able to learn the context as appropriate, i.e. as short or as long as is needed to discriminate between the different phonemes. The fact that the 3 frame recurrent architectures performed less well than the 7 frame non-recurrent architectures suggests that the context units were unable to represent a phonemic time course, or context, of even 4 frames (the difference in context between a 3 frame and a 7 frame time-window). Thus, to answer the questions asked above, we find that 1) recurrent LVQ, in this version, does not do as well with a small time-window as non-recurrent LVQ using a large time-window,



Squares of different sizes and colors represent different spectral coefficients. The size of each square represents magnitude; black indicates positive, white negative.

The sample shown at the top of the figure is a /k/ token, drawn from a multi-speaker database of 20 Japanese consonants.

The line of segments marked "seg=frame#" shows the speech segments obtained by shifting a 7 frame window over the token.

The bottom line of segments, marked "cv=frame#", shows the actual context activations corresponding to each frame number. Strong activations (large black squares) indicate that the reference vector corresponding to that coordinate was close to the previous input. The coordinates of the closest reference vector for the previous input are indicated outside the segment boxes.

The two lines of segments marked "rv category#, reference vector #" show the closest reference vector obtained for each speech segment. The top line shows the speech part of the reference vectors; the bottom line shows the context part.

Figure 2. Recurrent reference vector representations over time.

and 2) using time-windows of the same size, recurrent LVQ performs no better than non-recurrent LVQ.

What are the weaknesses of the version of recurrent LVQ proposed here? An examination of the context unit representation will help answer this question. Figure 2 shows what happens when a /k/ token is presented as input to the recurrent LVQ architecture. The speech input is shown, along with the first 6 speech segments obtained by scanning the time-window over the speech input, as well as the recurrently generated context units. Also shown are the speech and context parts of the closest reference vectors found over time. One can see that the speech part of the reference vectors closely resembles the speech input. One can also see that the context part of the reference vectors closely resembles the context unit activations. This is significant, as it shows that internal state is being represented in prototype form. Recurrent LVQ training generated context reference vectors that model, in prototype form, the activations of all the reference vectors at the previous time step.

However, if we look more closely at the context unit activations themselves, it appears that they reflect little more than the immediately prior state of the architecture. Context units whose activation is high (large black squares) reflect the fact that the corresponding reference vector was one of the closest vectors at the previous scan position of the time-window, but it does not appear that they reflect anything beyond that, i.e. further back in time. Thus, although the context reference vectors *were* able to learn something about the internal state of the recurrent architecture, it appears that the internal state reflects no more than the reference vector activations for the immediately preceding time step. The architecture as a whole was unable to generate context activations representing the whole time course of phoneme tokens.

This result seems to flow directly from the fact that the training is overly local, i.e. only extends one step back in time. Note that the context units themselves are recurrently generated; their values depend on the whole time course. However, it seems that the learning method we employed was unable to exploit this in the manner described Elman and McClelland in [6,7]. Note, however, that the tasks examined in [6,7] were quite different from the tasks we examined here. Elman examined the prediction of successive letters in sentences generated from a small grammar. McClelland et al. considered the learning of sequences of symbols generated by a finite state machine. These tasks are significantly "cleaner" than the task of processing actual speech data.

Our conclusion is that in order to achieve improvements over non-recurrent LVQ, we need to perform training over the whole time course of each training token, and not just at the level of each time-window position. This is the method used in [8,10]. It is more difficult to implement than the local, segmental recurrent training described here, but our results suggest that this might be necessary.

Conclusion

We here examined a form of recurrent training for LVQ architectures. The version proposed here is a simple attempt at creating an architecture that can learn to represent the appropriate degree of context needed to discriminate phonemes. This is more appealing than non-recurrent architectures which require that one fix the context. However, we found that the training method used here was overly local and failed to represent context recurrently. A more global training method, though more complex, is more promising in this respect. Visual examination of recurrently generated LVQ context units suggests that phonemic context might be representable in prototype-based learning algorithms.

References

[1] T. Kohonen; "Self-organization and Associative Memory" (2nd Ed.), pp. 199-202, Springer, Berlin-Heidelberg-New York-Tokyo, 1988.

[2] T. Kohonen, G. Barna and R. Chrisley; "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies," IEEE, Proc. of ICNN, Vol. I, pp. 61-68, July 1988.

[3] S. Katagiri, C.H. Lee, and B.H. Juang, "A Generalized Probabilistic Descent Method," Proc. of Acoustical Society of Japan, September 1990.

[4] E. McDermott and S. Katagiri, "Shift-Tolerant LVQ for Phoneme Recognition," Proc of IEEE Acoustics and Signal Processing, June 1991 [Forthcoming].

[5] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. Lang, "Phoneme Recognition: Neural Networks vs. Hidden Markov Models," Proc. of ICASSP, S3.3:107-110, April 1988.

[6] J.L. Elman, "Finding Structure in Time," *Cognitive Science*, 14, 179-211, 1990.

[7] A. Cleeremans, D. Servan-Schreiber and J. McClelland, "Finite State Automata and Simple Recurrent Networks." Neural Computation 1, 372-381, 1989.

[8] T. Robinson and F. Fallside, "Phoneme Recognition from the TIMIT database using recurrent error-propagation networks." Technical Report CUED/F-INFENG/TR.42, Cambridge University Engineering Department, March 1990.

[9] J. Makhoul, S. Roucos, and H. Gish, "Vector Quantization in Speech Coding". Proc IEEE 73, No. 11, 1551-88.

[10] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, edited by D.E. Rumelhart and J.L. McClelland (MIT Press, Cambridge, USA, 1986)

[11] E. McDermott, S. Katagiri, "Shift-Tolerant Phoneme Recognition Using Kohonen's LVQ2." Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing, 1989.

[12] E. McDermott, "LVQ3 for Phoneme Recognition." Proc. of Acoustical Society of Japan, Spring 1990.

[13] Y. Minami, T. Hanazawa, H. Iwamida, E. McDermott, K. Shikano, S. Katagiri, M. Nakagawa, "On the Robustness of HMM and ANN Speech Recognition Algorithms." Proc. of International Conference on Speech and Language Processing, Kobe, Japan, 1990.