Internal Use Only
非公開

TR－A－0059

0018

LVQ-Based Shift-Tolerant Phoneme Recognition

－LVQによる時間シフトに強い音韻認識－

Erik MCDERMOTT and Shigeru KATAGIRI
エリック・マクダモット　片桐　滋

1989. 8. 22

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

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

LVQ-Based Shift-Tolerant Phoneme Recognition

Erik McDermott and Shigeru Katagiri

ATR Auditory & Visual Perception Research Laboratories,
Sanpeidani, Inuidani, Seika-cho, Soraku-gun,
Kyoto 619-02, Japan

Abstract

In this paper we describe a shift-tolerant neural network architecture for phoneme recognition. Our system is based on algorithms for LVQ (Learning Vector Quantization) [1, 2], recently developed by Teuvo Kohonen, which pay close attention to approximating optimal decision lines in a discrimination task. Recognition performances in the 98-99% correct range were obtained for LVQ networks aimed at speaker-dependent recognition of phonemes in small but ambiguous Japanese phonemic classes. A correct recognition rate of 97.7% was achieved by a single, larger LVQ network covering all Japanese consonants. These recognition results are at least as high as those obtained in the Time Delay Neural Network system developed by Alex Waibel et al. (1988), and suggest that LVQ could be the basis for a successful speech recognition system.

Contents

List of Illustrations

List of Tables

1. Introduction

To achieve the end goal of large vocabulary, speaker-independent, continuous speech recognition, it seems that a useful first step is to achieve high-performance phoneme recognition. For a system to yield this high performance, it must be endowed with high classification power and invariance under translation in time. The Time Delay Neural Network architecture recently developed by Alex Waibel et. al. [3] and the Back-propagation algorithm used therein seem to possess these properties, and very high recognition results were attained using this system. However there are other algorithms with these capabilities. Recent work by Kohonen [2], suggests that Learning Vector Quantization (LVQ) is a slightly more powerful classifier than Back-propagation, and furthermore that LVQ requires significantly less learning time than either Back-propagation or the Boltzmann Machine. Thus LVQ seems like a viable candidate for tasks that involve a very large training set size, such as speech recognition. We here present a shift-tolerant, LVQ-based phoneme recognition system which is capable of attaining recognition rates that are as high as those obtained for the TDNN system, and which requires little training time.

2. Description of the Learning Vector Quantization Algorithms

Kohonen presents two versions of LVQ [2]; here we refer to them as LVQ1 and LVQ2. In both versions, each category to be learned is assigned a number of reference vectors, each of which has the same number of dimensions as the input vectors of the categories. In the recognition stage, an unknown input vector will be categorized by finding the reference vector that is closest to that input vector. The category that the reference vector belongs to will be given as the categorization of the unknown input vector. This classification scheme means partitioning the vector space into regions, or cells, defined by individual reference vectors. Both LVQ1 and LVQ2 assume a good initial configuration; this can be obtained by using the traditional K-means clustering procedure [7]. This initial configuration will not be optimal, but it places the reference vectors in the right general position. LVQ training then adjusts these positions so that each input vector has a reference vector of the right category as its closest reference vector. The difference between LVQ1 and LVQ2 is that they each use a different way of selecting the reference vectors to be adapted. We now describe each algorithm.

2.1. LVQ1

The adaptation rule for LVQ1 is as follows. For each trial, an input vector from one of the categories is presented as the input to the system. All the reference vectors are searched and the reference vector closest to the input vector is found, using a Euclidean measure of distance. If this reference vector belongs to the same category as the input vector, it is moved closer to the input vector, in proportion to the distance between the two vectors. If the closest reference vector belongs to a category other than that of the input vector, it is moved away, again in proportion to the distance between the two vectors. One can see how this procedure is related to the goal of having a vector of the correct category as the reference vector closest to each input vector. Learning here is supervised: one needs to know the correct categorization of the input vector during training.

More specifically, the adaptation rule for a closest reference vector $m_i \in R^n$ and an input vector $x \in R^n$ is

$$m_i(t+1) = m_i(t) + \alpha(t)\,(x(t) - m_i(t)),$$

if the reference vector belongs to the same category as the input vector, and

$$m_i(t+1) = m_i(t) - \alpha(t)\,(x(t) - m_i(t))$$

if the reference vector belongs to a different category. Here t is the discrete time index, and $\alpha(t)$ is a monotonically decreasing function of time.

2.2. LVQ2

LVQ2 requires that a number of conditions be met for vector adaptation to occur. These conditions allow the system to pay closer attention to the decision lines of a given categorization problem. This, it seems, accounts for the superior performance of LVQ2.

Kohonen, in his formulation of the LVQ2 algorithm, is particularly concerned with the problem of approximating decision lines corresponding to the optimal Bayes decision rule. Given a vector x, a set of classes $\{C_i, i=1,2,....,K\}$, the probability density function $p(x \mid C_i)$ of x in a class C_i, and the a priori probability $p(C_i)$ of a class C_i, and the distribution $d_i(x) = p(x \mid C_i)p(C_i)$ of class C_i, the Bayes decision rule is as follows:

x is assigned to C_i iff $d_i(x) > d_j(x)$ for all $j \neq i$

This rule will minimize the number of misclassifications [2, 11]. This becomes particularly relevant when the class distributions overlap. If there is overlap, it is impossible to separate the classes perfectly; the task becomes that of finding the decision line which minimizes the number of misclassifications. This will be achieved by a decision line at the place where the class distributions cross. Ideally, a neural network should generate decision lines that approximate this optimal line. This is the motivation for LVQ2.

Figure 1 helps to illustrate vector adaptation in LVQ2, for a simple one-dimensional situation. For a given training vector x, three conditions must be met for learning to occur: 1) the nearest class must be incorrect; 2) the next-nearest class (found by searching the reference vectors in the remaining classes) must be correct; 3) the training vector must fall inside a small, symmetric window defined around the midpoint of the reference vectors m_i and m_j-- this midpoint (in higher dimensions, "midplane") being the decision boundary effected by the two vectors. If these conditions are met, the incorrect reference vector is moved further away from the input, while the correct reference vector is moved closer, according to:

$$m_i(t+1) = m_i(t) - \alpha(t)\,(x(t) - m_i(t))$$
$$m_j(t+1) = m_j(t) + \alpha(t)\,(x(t) - m_j(t))$$

where x is a training vector belonging to class j, m_i is the reference vector for the incorrect category, m_j is the reference vector for the correct category, and $\alpha(t)$ is a monotonically decreasing function of time.

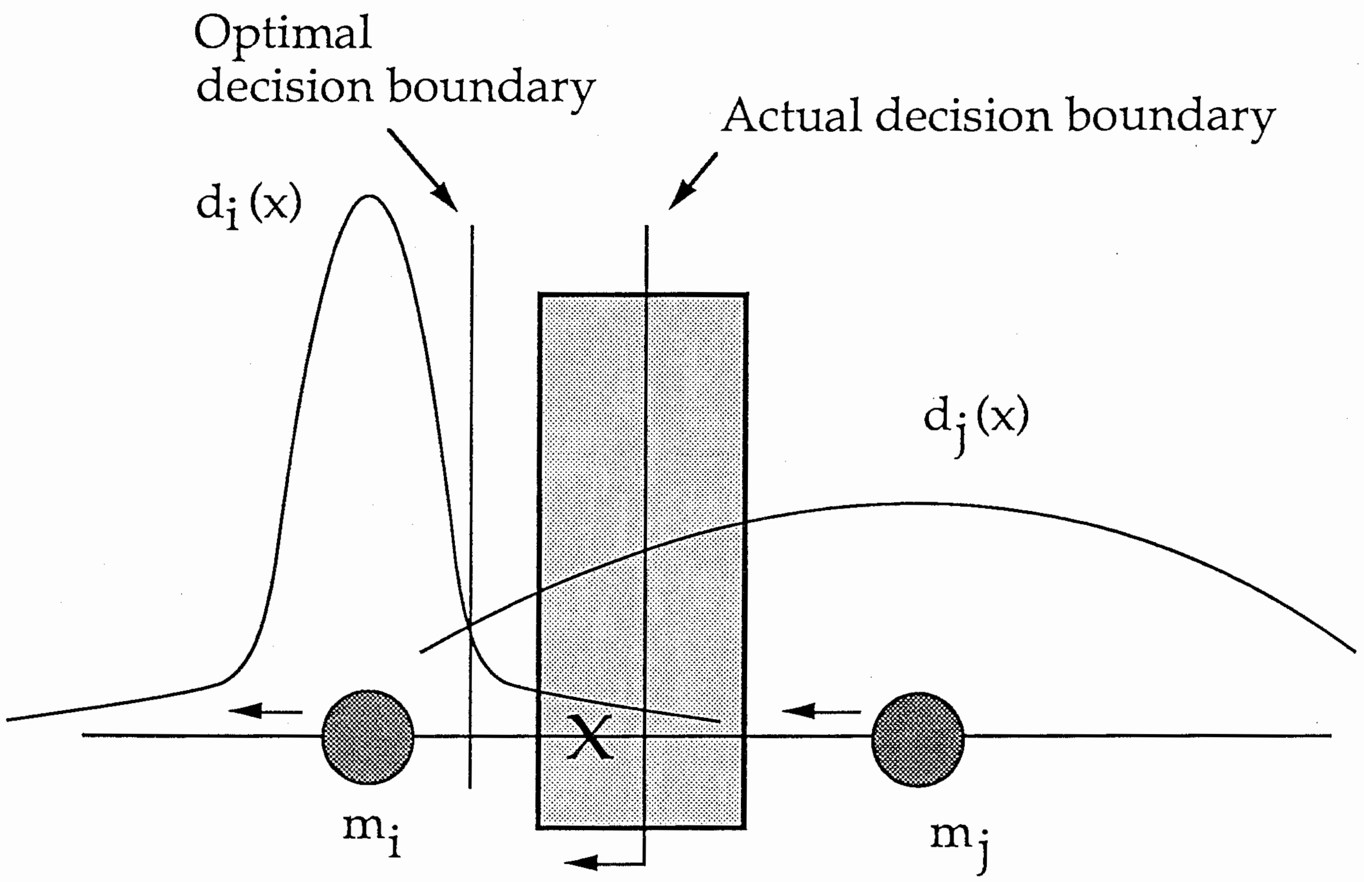
These requirements, taken together, assure that the decision line between the two vectors will eventually attain a near-optimal position given the probability distributions of the categories, namely, the place where the distributions cross.

The intuition here is that LVQ2 is making use of local difference in class distributions to move the boundary in the right direction. This local difference in class distributions is measured indirectly, by measuring a difference in the number of misclassifications on either side of the LVQ2 window. Figure 1 may help explain the overall, stochastic effect. A difference in the class distributions at the position of the actual boundary in the pattern space is taken to indicate that the actual boundary is not optimal. This is the case in Figure 1, where at the position of the actual boundary, the distribution of class j is greater than that of class i. Thus, there will be more misclassifications of j's on the left side of the window than misclassifications of i's on the right side of the window. Now note that every misclassification of a j token on the left side of the window satisfies the LVQ2 conditions. LVQ2 will then push the closest but incorrect m_i reference vector away from those j's, while pulling the m_j reference vector closer. Precisely the opposite vector motions will occur for misclassifications of i's on the right side of the window. Of course, these vector motions displace the actual boundary realized by these vectors. In the Figure 1 scenario, there will then be more misclassified j's pushing the actual boundary to the left than misclassified i's pushing the boundary to the right. The net effect, then, will be to move the actual boundary to the left, i.e. closer to the optimal boundary. This will continue until the number of j misclassifications on the left side of the window is equal to the number of i misclassifications on the right side of the window; for a small window, this position will be close to the optimal Bayes boundary.

To define the window in higher dimensions, we used the following method. The distances d_i and d_j from x to m_i and m_j are calculated, and the ratio d_i/d_j is found. Clearly, if m_i is the closest vector, this ratio will always be less than 1. The window is then defined by setting a lower limit L for this ratio. Only training vectors with a d_i/d_j ratio greater than L (and less than 1) will be considered inside the window. The closer L is to 1, the smaller the window. Setting this kind of limit amounts to defining the window as the space between two Apollonian hyperspheres, each surrounding one of the two reference vectors [1].

Figure 1 also helps to illustrate the difference in goals between LVQ and K-means clustering. The reference vector configuration presented in Figure 1 more or less corresponds to what the result of K-means clustering might be, given the category distributions shown. One can see, however, that the decision line realized by these two reference vectors is quite far from the optimal Bayes decision boundary, which is the target of LVQ learning.

Goal: optimal discrimination of categories i and j



x: a training vector of category j

m_i, m_j: reference vectors

$d_i(x)$, $d_j(x)$: category distributions

Figure 1. LVQ2 adaptation, one dimensional case.

To best comply with LVQ2 theory, the window should be as small as possible given the size of the training set. For an infinite training set, the window could be infinitely small. For finite training sets, there is the danger that an overly small window will be stuck between the training vectors it excludes, in a non-optimal position. But on the whole, setting the window size seemed to pose few problems; very similar performances were found for a broad range of window sizes.

In both LVQ1 and LVQ2, $\alpha(t)$ is a small number which decreases over time. In our simulations, we usually used a linearly decreasing function of time such as

$$\alpha(t) = \alpha(0)\,(1 - t/M),$$

where M is the maximum number of iterations, after which learning is halted.

As such a function will force convergence, we need to estimate the number of iterations before training begins. Lacking a theory for this, we determined this number experimentally, using recognition rates on test data as the criterion for good learning. One way of estimating $\alpha(0)$ is to choose it as high as possible without it giving rise to an unstable system. In our system, a typical value of $\alpha(0)$ is 0.1; a typical value of M is 10 times the number of training tokens. We usually performed several training runs on the same set of reference vectors, progressively lowering the value of $\alpha(0)$.

To allow for the approximation of the optimal boundary, it is important that the training tokens be presented in proportion to the overall probabilities (i.e. P(c)'s) of their classes. The training procedure, for each trial, is then to select a training token at random from the whole training set. Tokens from seldomly occuring categories are not duplicated; this would artificially raise the class distributions, and prevent the system from approximating the optimal class boundaries.

3. System Architecture

This phoneme recognition system was previously described in [5] and [6]. Figure 2 shows the architecture of the recognition system used for the /b/, /d/, /g/ task. Each category is assigned a number of reference vectors. The LVQ training procedure is then applied to speech patterns that are stepped through in time, thus providing the system with a measure of shift-tolerance. To achieve this effect, we defined a 7 frame window[1] which is shifted, one frame at a time, over the 15 frame speech token. Each window position yields an input vector of 112 dimensions (7 frames x 16 channels). Given this input vector, LVQ1 or LVQ2 is applied as described above.

This moving window scheme requires a slightly different recognition procedure than simply finding the closest vector, as there are now several closest

[1] The choice of 7 frames here is somewhat arbitrary; widths ranging from 6 to 9 frames gave very similar performances. Undoubtedly the optimal width will vary from task to task.

vectors, one for each window position. For each window position we calculate the distances between the input vector and the closest reference vector within each category. From this distance measure, each category is assigned an activation value that is high for small distances, low for large distances:

$$A(c, t) = 1 - \frac{d(c)}{d(0) + d(1) + d(2)}$$

After the window has been shifted over all 15 frames, the activations obtained at each window position are summed, for each category. The category with the highest overall activation is chosen as the recognized category. Note that these activation calculations are not done using additional layers of connections: our system is essentially a one layer system.

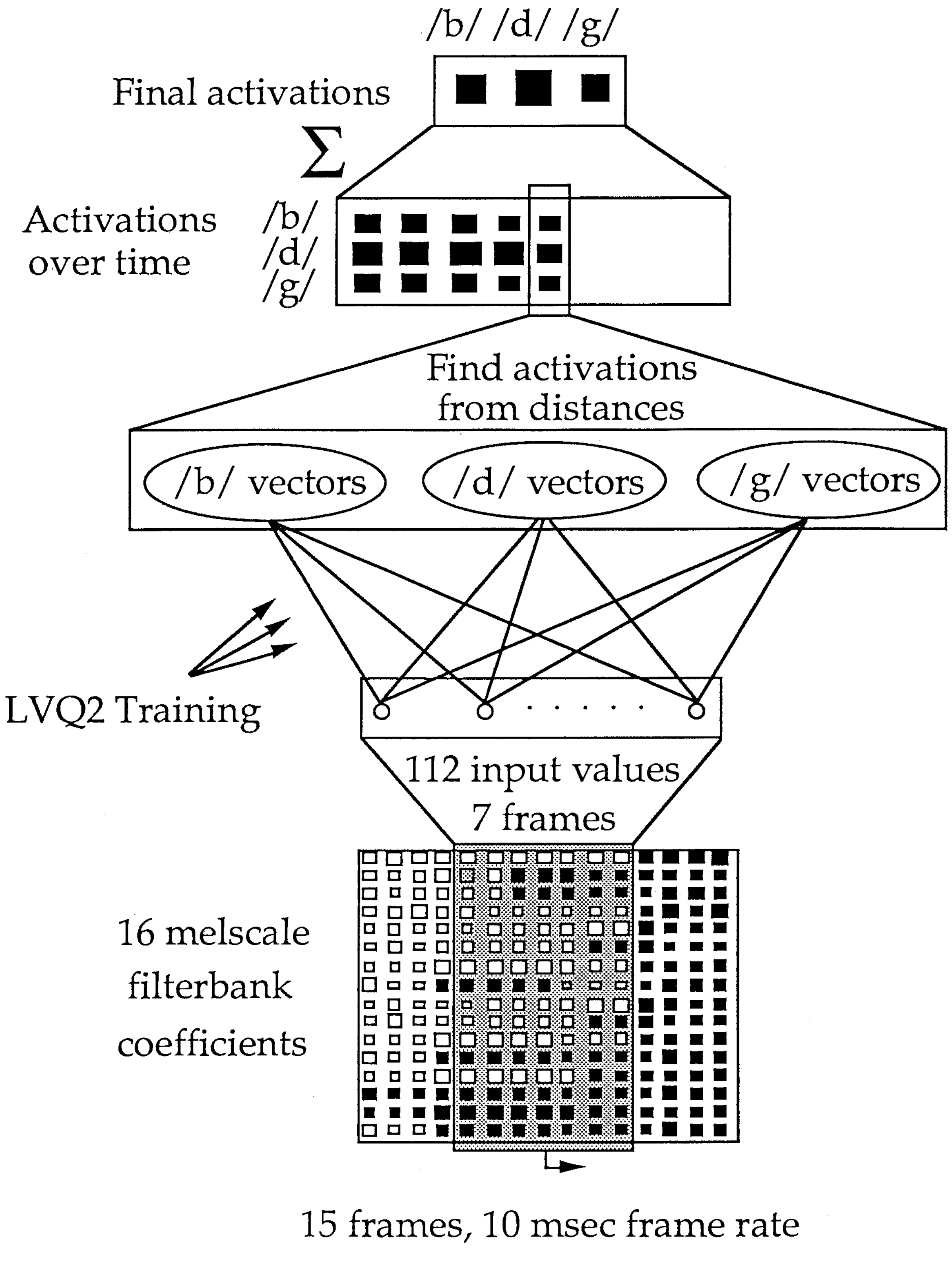


Figure 2. System architecture, /b/, /d/, /g/ task

4. Experiments

4.1. Speech Data Representation and Database

To be fair in our comparison with the TDNN system, we used exactly the same data representation as that used in TDNN . One phoneme token consisted of 15 time frames of 16 melscale spectrum channels, with a frame rate of 10 msec. Input speech was sampled at 12 kHz, hamming windowed, and a 256-point FFT computed every 5 msec. Melscale coefficients were generated from the power spectrum and coefficients adjacent in time were collapsed, yielding an overall frame rate of 10 msec. The coefficients were then normalized between 1.0 and -1.0 with the average at 0.0. Figure 2 displays these coefficients as black or white squares of varying sizes, size representing magnitude, black for positive values, white for negative values.

These tokens were drawn from a database of about 5230 common Japanese words, uttered in a sound-proof booth by a male professional announcer. This is the same database as used in the TDNN experiments. The database was split into a training set and a testing set of 2615 utterances each, from which the phoneme tokens were then extracted using manually selected acoustic-phonetic labels. For consonant tokens, the center frame was set at the border between the consonant and the following vowel. For vowels, the center frame of each token was set at the center position of the vowel. This extraction procedure is identical to that used in the TDNN experiments. In effect, the tokens we used for both training and testing of the system were identical to those used in TDNN.

4.2. System Configuration

One of the reasons LVQ2 learns fast is that the initial conditions start out fairly well. Whereas Back-propagation networks are typically initialized by setting the weights to small random values, the reference vectors in LVQ2 can be initialized by performing K-means clustering, as mentioned above. The method most in line with LVQ theory, and recommended by Kohonen [1], would be to perform K-means clustering on the training tokens for all categories, and then assign category labels to the resulting reference vectors. A faster and simpler method is simply to perform K-means clustering on the training tokens for just one category at a time. This method was found by Yokota et al.[8] to provide an initial system performance that is as good or better than that resulting from a variety of initialization methods, including the first method described here. We should mention, though, that performance after LVQ training is not significantly altered by these different initialization methods. Even very crude methods, such as using randomly selected training tokens to initialize the reference vectors, produce fairly good initial performance, and very high performance after LVQ training. In sum, adequate initial conditions can easily be found for LVQ reference vectors. These initial vector positions are not the optimal positions, but they place the reference vectors in the right general area. This, it seems, plays a key role in reducing training time.

One question is how to determine the appropriate number of reference vectors. What should the total number of vectors be, and how many of these vectors should be assigned to each category? As to the first question, it seems clear that a large number of reference vectors can in general only help performance. However, there is a point at which the performance no longer increases significantly (or at all) with increases in the number of reference vectors. By setting the number of reference vectors such that the system is at this point of diminishing returns, we retain near-optimal performance while keeping the number of reference vectors low.

As to the second question, how many reference vectors should be assigned to each category, the theoretically sound method is perhaps to assign vectors in proportion to the class probabilities, P(c); this is the method reccomended by Kohonen [1]. However, we have found that a broad range of choices leads to essentially the same performance.

4.3. Recognition of /b/, /d/, /g/

As a first step in the evaluation of LVQ1 and LVQ2, we applied our system to the /b/, /d/, /g/ task. For this task we were able to obtain an overall recognition rate of 99.2% for 658 testing tokens (open test) from one speaker.

We implemented our system on an 8-processor Alliant super mini-computer. The simple vector operations that constitute the core of LVQ allowed for very easy parallelization and thus high learning speed. Furthermore, the initial conditions start out well, so the required number of trials is not particularly large. Thus, for LVQ2 on the /b/, /d/, /g/ task, a recognition rate of 98 ~ 99% can be attained in about one minute of CPU time, corresponding to 5 epochs of training (one epoch = one full presentation of the training set).

4.4. Evaluation of K-means, LVQ1 and LVQ2, on the /b/, /d/, /g/ task

K-means clustering is a very powerful method for vector quantization, and it provides an effective way of initializing LVQ reference vectors. However, the goal of K-means is not to reduce the number of misclassifications in a discrimination task, but to reduce the average distortion of a vector quantizer. The two goals may overlap, but are not identical; achieving one of these goals may well be at the expense of the other. To investigate this issue, we considered the difference in classification performance between LVQ and K-means clustering, on the /b/, /d/, /g/ task, for different numbers of reference vectors ranging from 15 to 300. Here each category was allotted reference vectors in proportion to the number of training samples available for that category. For each point in the plot, K-means was run on four different sets of initial reference vectors (sampled at random from the training set). Using the best of these four runs as the starting configuration, both versions of LVQ learning were performed. For a given initial configuration, LVQ2 was performed just four times, with exactly the same parameter settings, the only difference thus being the order of presentation of training tokens. LVQ1, on the other hand, required more exploration of the parameter space to improve on the performance of the initial configuration. Figure 3 shows, for each choice of total number of reference vectors, the test data performance for the best of the four K-means runs, and for the best of the LVQ training runs. This figure illustrates the previosuly mentioned asymptotic behavior of LVQ as the number of reference vectors is increased. We also see that LVQ nearly always does better than K-means, and that the difference in performance is accentuated for small numbers of reference vectors. These aspects are particularly visible for LVQ2. To achieve a performance of around 98%, 9 reference vectors are sufficient for LVQ2, while about 175 are necessary for K-means.

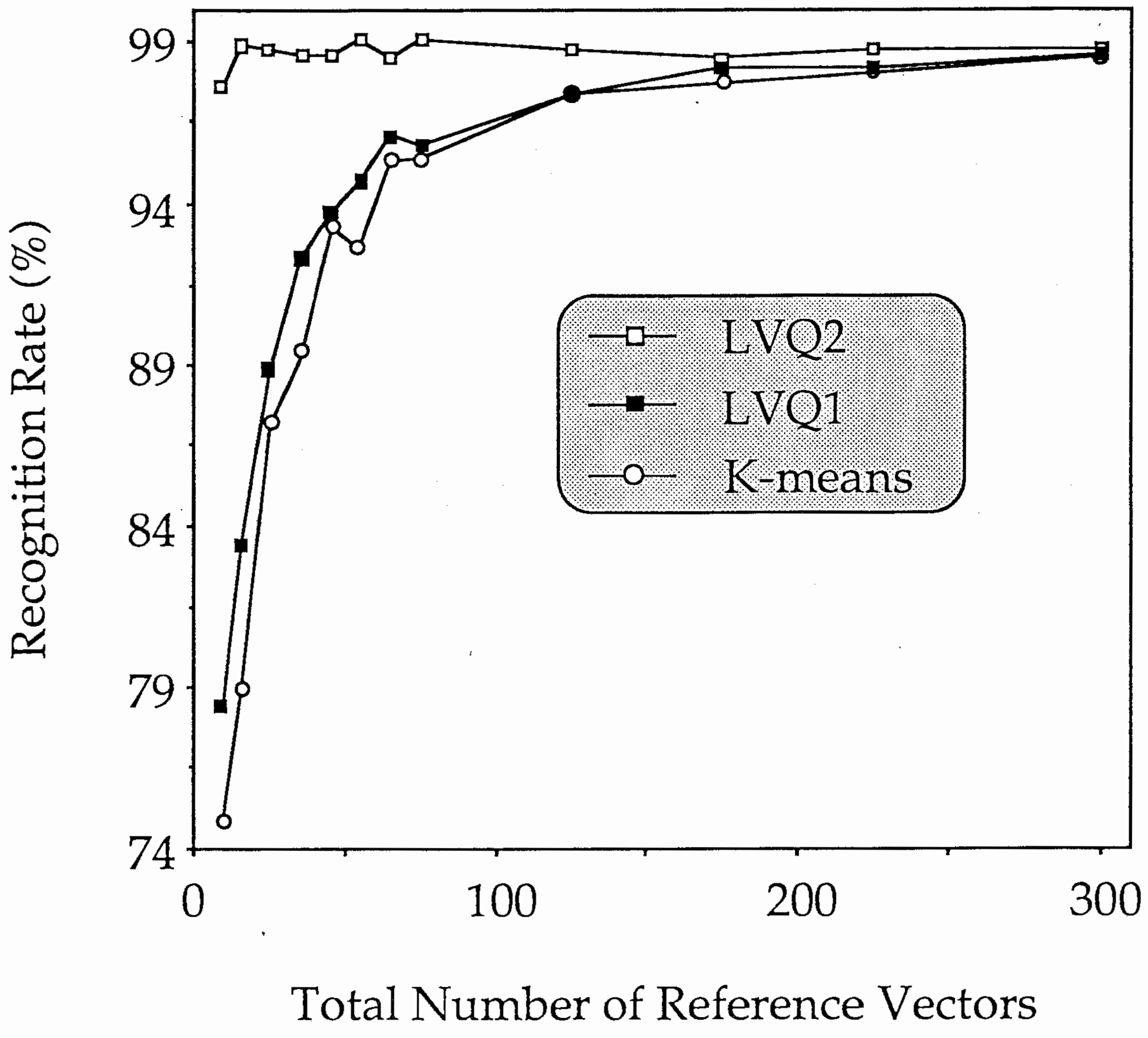


Figure 3. Performance vs. number of reference vectors, bdg task.

4.5 Recognition of Phonemes in Other Phonemic Classes Using LVQ2.

Encouraged by these results for /b/, /d/ and /g/, we then applied our LVQ2 architecture to additional phoneme classes: the unvoiced stops, /p/, /t/, /k/; the nasals /m/, /n/, and syllabic nasals; the fricatives /s/, /sh/, /h/, /z/; the affricates /ch/ and /ts/; the liquids and glides /r/, /w/, /y/; and the vowels /a/, /i/, /u/, /e/, /o/. Together, these constitute nearly the entire phoneme set for Japanese. The training set size for each of these classes was roughly the same size as the test set size, shown in Table 1.

LVQ2 networks for each of these phonemic classes were initialized using K-means clustering and then trained in the same manner as described above.

The test data recognition results for these phonemic classes are shown in Table 1. Here we are comparing our results with those of the TDNN system [3], [4]. Note that each network here is only trained to discriminate phonemes within one phonemic class, and thus does not know about phonemes from other classes.

As can be seen, we obtained recognition results that are at least as good as those obtained in the TDNN system. Furthermore, all of these tasks required relatively little training time in the LVQ2 system. We do have a problem with the phoneme /p/: due to the very infrequent occurence of /p/'s in Japanese, our training data is not sufficient to produce good generalization.

Most of these networks did not achieve perfect recognition of the training data. The training data performance was typically in the high 99% range.

task	LVQ2			k-means	TDNN
	#errors/ #tokens	%correct	total %	total %	total %
b	2/227	99.1	99.2	78.7	99.0
d	0/179	100			
g	3/252	98.8			
p	6/15	60.0	98.9	95.7	98.7
t	0/440	100			
k	5/500	99.0			
m	4/481	99.2	98.8	83.7	96.6
n	7/265	97.4			
N	4/488	99.2			
s	4/538	99.3	99.4	98.8	99.3
sh	0/316	100			
h	0/207	100			
z	3/115	97.4			
ch	0/123	100	100	100	100
ts	0/177	100			
r	0/722	100	99.6	99.2	99.9
w	1/78	98.7			
y	3/174	98.3			
a	0/600	100	99.1	96.7	98.6
i	2/600	99.7			
u	14/600	97.7			
e	6/600	99.0			
o	4/600	99.3			

Table 1. Test data recognition rates for small phonemic tasks.

4.6 Recognition of All Japanese Consonants Using LVQ2

We next built an LVQ2 network that can discriminate among all Japanese consonants, not just within small phonemic classes. This network, illustrated in Figure 4, has the very same architecture as the small networks above, but with more categories. As before, the network is initialized using K-means clustering, and then trained using LVQ2.

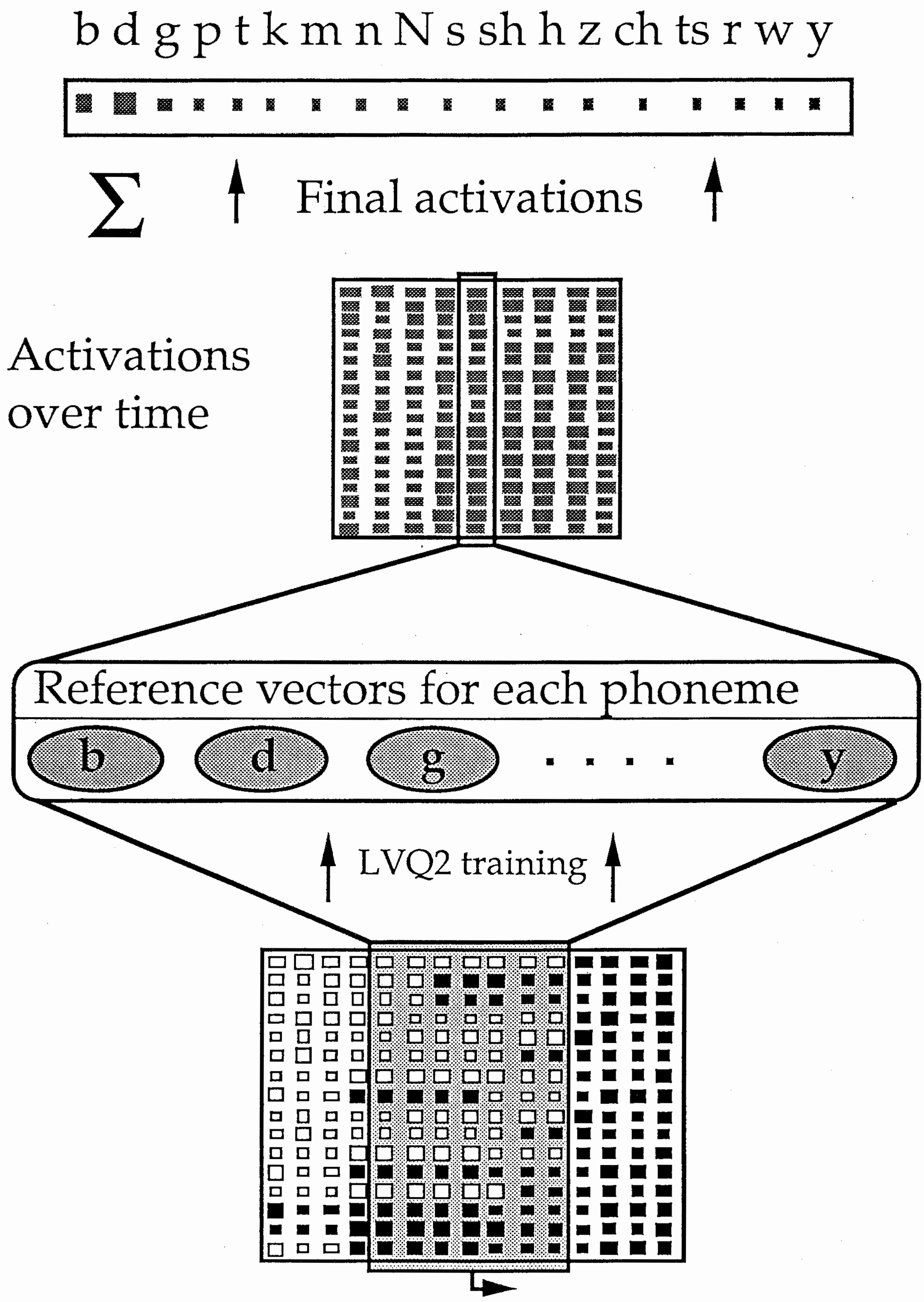
First, to compare this LVQ2 network with its TDNN counterpart [4, 9], we used the very same training and testing sets used in TDNN. The training set consisted of 5,063 tokens; the testing set consisted of 3061 tokens. For these data sets, our system architecture, initialized using K-means, recognizes 92.4% of the test data correctly; after LVQ2 training, the overall performance is 97.1%. This is compared with the best TDNN result for the all-consonant task, 96.7%.

Next, we trained and evaluated the all-consonant LVQ2 network using larger data sets. In TDNN, for reasons that we will not discuss here, the data sets for very frequently occurring phonemes were limited to 500 or 600 tokens (e.g. for k and the vowels). To be fair in our comparison, all the results presented so far used the same limited data sets. However, our purpose here is not solely comparison with TDNN, and we wanted to see how an all-consonant LVQ2 network performs using all the available tokens for each phoneme. Accordingly, our next step was to train an LVQ2 network using a full training set of 5,973 tokens, and evaluate it with a full testing set of 5,960 tokens. As expected, performance increases with additional training data. The overall performance here is 97.7%.

These results are summarized in Table 2. As to recognition of training data, for the first network, trained on the limited data set, the performance was 99.3%, while for the network trained on the full data set, the performance was 99.4%.

Training each network here required about 10 ~ 30 epochs in all, resulting in about 2 or 3 hours of CPU time: substantially more than for the small networks but still reasonable.

For the all-consonant results presented here, each category was assigned 25 reference vectors. Our experience suggests a picture very similar to that detailed above for the /b/, /d/, /g/ network: above a certain total number of reference vectors, performance ceases to improve.



b d g p t k m n N s sh h z ch ts r w y
Σ
Final activations
Activations over time
Reference vectors for each phoneme
b
d
g
. . . .
y
LVQ2 training

Figure 4. System architecture, all consonant task

Data Set	LVQ2	K-means	TDNN
Limited	97.1%	92.4%	96.7%
Full	97.7%	91.5%	-

Table 2. Recognition rates for all-consonant networks.

5. Comparison with other LVQ Architectures

To provide additional motivation for the architecture described and tested above, we here describe some of our findings for other architectures. First, two architectures that are not trained on time-shifted speech segments will be described and their performances reported. This is to suggest that training on shifted speech segments really does improve performance by adding a measure of shift-tolerance to the system. Second, we describe an architecture that is shift-tolerant but uses a different recognition rule from the one described in Section 3. This is to note the importance of decision rules that integrate information over time. These architectures were evaluated on the all-consonant recognition task using the full training and testing sets mentioned above. The studies shown here are not intended to be exhaustive descriptions of all the possible architectures, but rather to illustrate the key features of the architecture we are focussing on in this paper (in the following sections, the "main" architecture.)

5.1. Two Shift-Sensitive Architectures

One very simple alternative to training the system on shifted speech segments is simply to train using input vectors generated from the whole token, i.e. vectors of 240 dimensions, corresponding to the full 15 frames of 16 FFT coefficients. This is the basis for the first shift-sensitive architecture, illustrated in Figure 5. As above, each category was associated with a number of reference vectors, and the LVQ training procedure was applied to these 240 dimensional training vectors. Recognition is simply done by presenting an input vector and finding the class of the closest reference vector.

Twelve reference vectors were assigned to each class, corresponding to roughly the same number of parameters as used in the shift-tolerant system for all-consonant recognition, which used 25 reference vectors for each class, but where the vectors had fewer dimensions. As above, K-means was performed, resulting in an initial test data performance of 90.4% for the all-consonant task. Trained on the full data set for the all-consonant task, this architecture yielded a performance on training data of 98.9%, and a test data performance of 94.7%. This is in contrast to the main architecture, which had a training data performance of 99.4% and a test data performance of 97.7%, as described above. Thus in comparison to the main architecture, this shift-sensitive architecture suffers from a much greater drop between performance on training data and performance on test data.

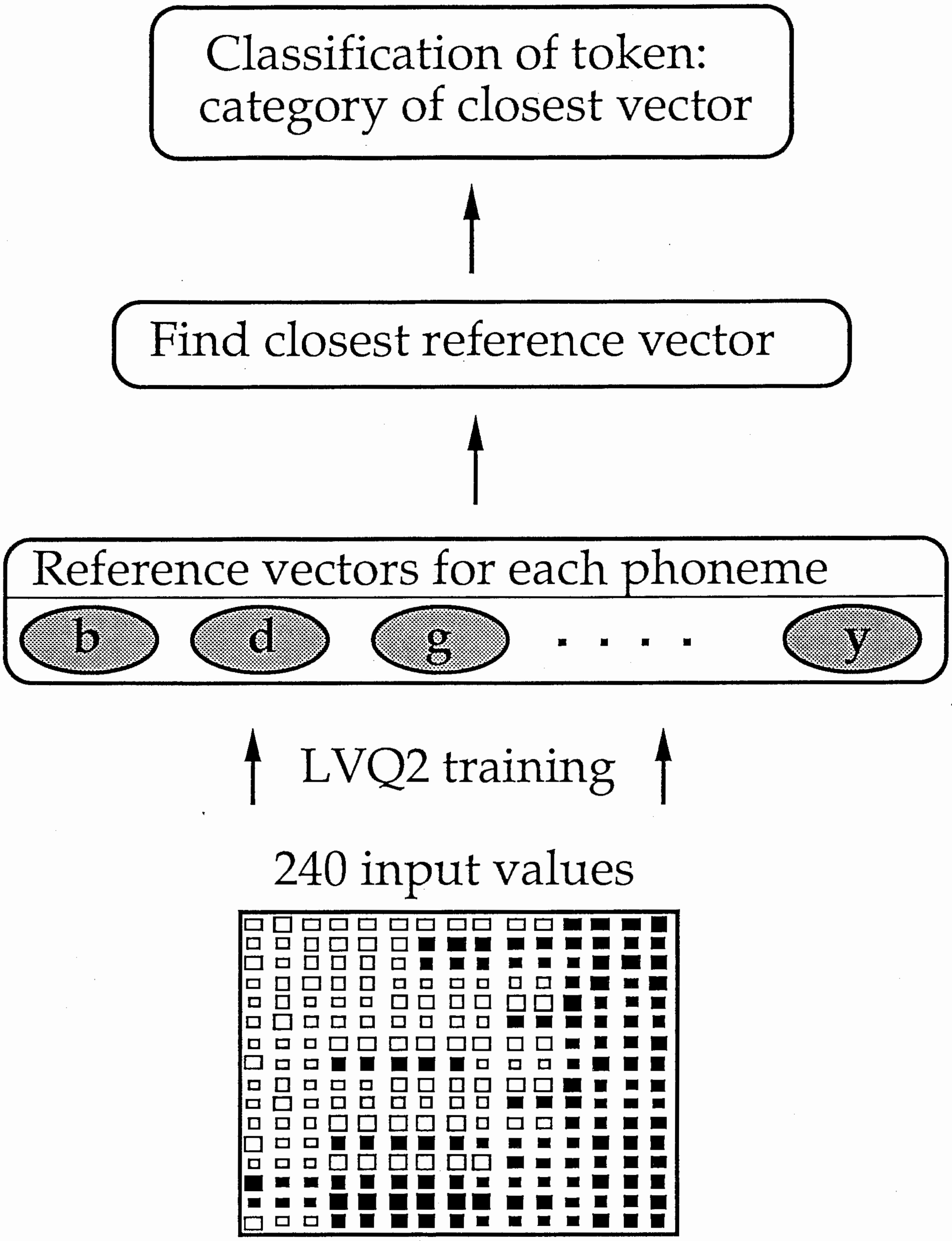

Classification of token: category of closest vector
Find closest reference vector
Reference vectors for each phoneme
b
d
g
. . . .
y
LVQ2 training
240 input values

Figure 5. First shift-sensitive architecture

The drop in test data performance just described seems clearly related to the shift-sensitive nature of the architecture. However, it is conceivable that the difference in performance, compared to that of the main architecture, is due to the difference in dimensionality (112 dimensions in one architecture, 240 in the other), and not to an inherent advantage in training on shifted speech segments. To try to separate the effect of dimensionality from that of training on time-shifted patterns, we examined the effect of using essentially the same shift-sensitive architecture, but this time limiting the token size to 7 frames-- i.e. the same window width as in the main architecture. These short tokens were taken from the central 7 frames of the usual 15 frame tokens. Since the consonant tokens have been hand-aligned such that the transition point between consonant and vowel occurs close to the center position, these 7 frame tokens contain very useful information concerning the token's phonemic identity.

This network is shown in Figure 6. As above, we used the same number of parameters as for the main architecture's all-consonant net, i.e. 25 reference vectors per class. K-means initialization resulted in 91.6% recognition performance on test data; LVQ2 training resulted in training data performance of 99.2%, and test data performance of 93.8%. Once again, compared to the shift-tolerant architecture, we see a significantly bigger drop between performance on training data and performance on test data.

For both shift sensitive architectures described here, increasing the number of reference vectors did not improve performance significantly.

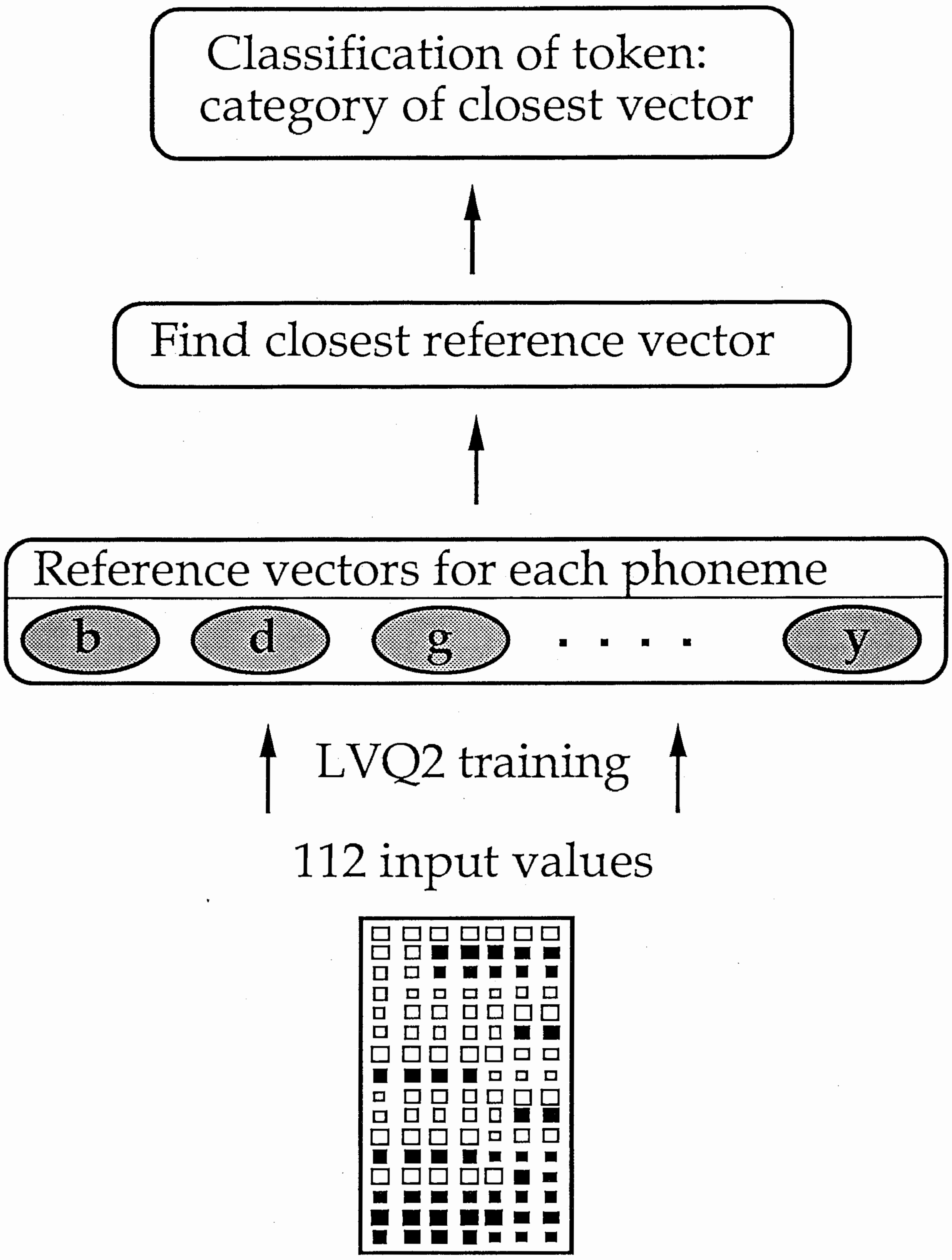

Classification of token: category of closest vector
Find closest reference vector
Reference vectors for each phoneme
b
d
g
. . . .
y
LVQ2 training
112 input values

Figure 6. "Center only" shift-sensitive architecture.

5.2. A Different Recognition Rule

Aside from training reference vectors on shifted speech segments, the other salient feature of the main architecture is the recognition rule. In the recognition phase, as described above, we shift the 7 frame window over the token, generate category activations for each position of this window on the token, and sum activations over time to generate final activations for each category. The particular method we employed to integrate information over time is just one choice out of many possible choices; examination of a variety of similar recognition rules revealed essentially no differences in performance. The key feature seems to be that the recognition rule integrates information over all the positions of the time-shifted window. To illustrate this point, we here describe a recognition rule that does not integrate information over time in this way, and which thus leads to a drop in performance.

The basic recognition rule in LVQ is simply to find the nearest neighbour among all the reference vectors. In this spirit, one possibility for a recognition rule in our phoneme recognition system is as follows. First, shift the window over the token, recording, for each position, the distances between input vectors and closest reference vectors. Once the window has been fully shifted over the token, choose, among all these distances, the smallest one, and the reference vector corresponding to it. The category of that reference vector alone is then given as the categorization of the whole token. This network is shown in Figure 7.

To test this rule, we used the same set of reference vectors that generated the 97.7% test data recognition on the all-consonant recognition task above, using the main shift-tolerant architecture. No additional training was performed on these vectors, they were used only in the recognition phase, during which the new rule was used instead of the summation of activations. Applied to the all-consonant task, 25 reference vectors per class, this recognition rule yielded a performance of 84.5% after K-means initialization. Performance after LVQ2 training was 98.8% for training data and 95.0% for test data. Here again, we see a relatively larger drop between performance on training data and performance on test data, compared to the main architecture presented here.

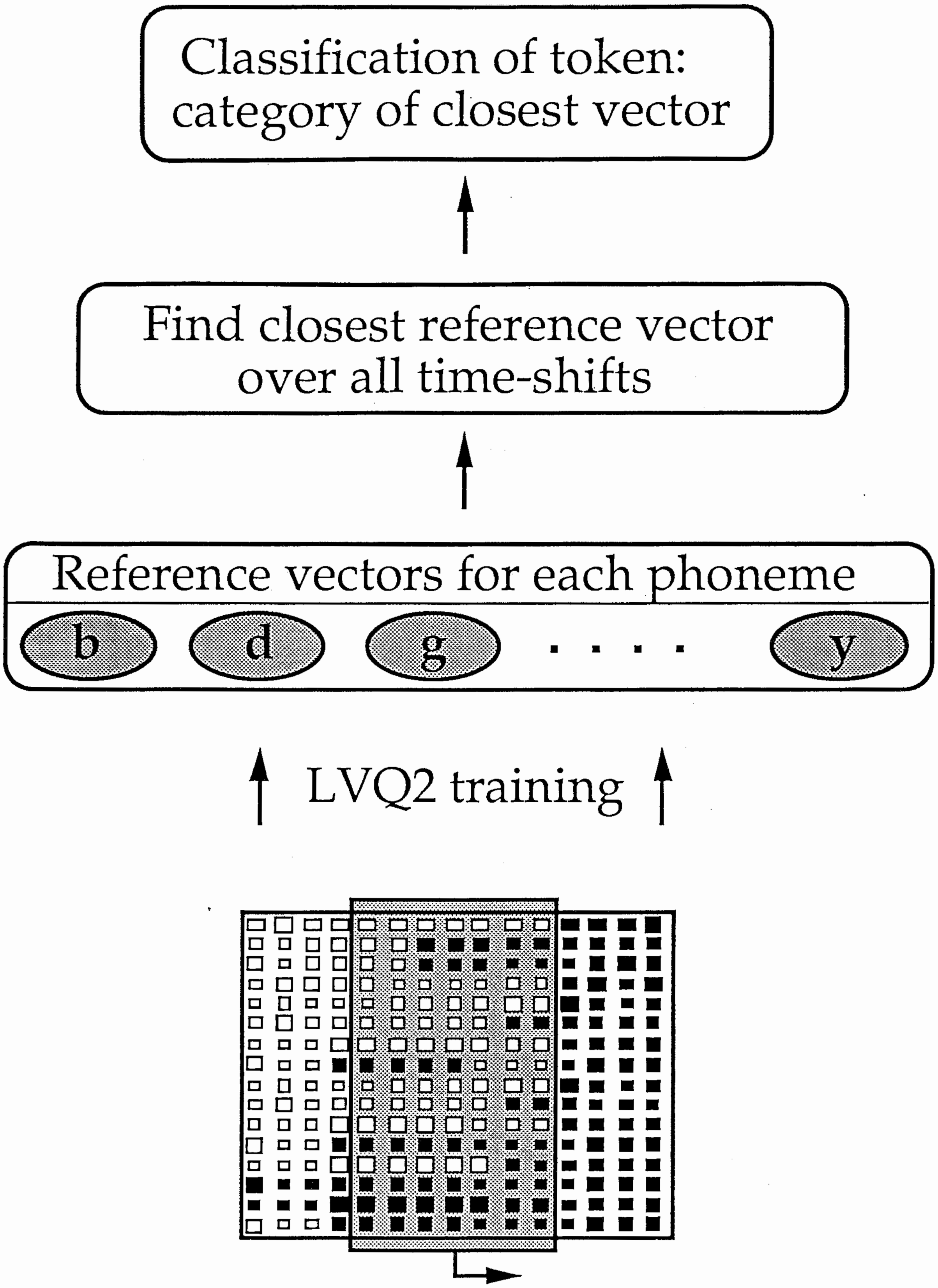

Classification of token: category of closest vector
Find closest reference vector over all time-shifts
Reference vectors for each phoneme
b
d
g
. . . .
y
LVQ2 training

Figure 7. Architecture with a different recognition rule.

6. Discussion

Our results suggest that LVQ, particularly in its LVQ2 version, is an extremely powerful classifier, comparable or even slightly superior in ability to Back-propagation.

Clearly, *from the viewpoint of classifier performance,* both LVQ1 and LVQ2 are more powerful than K-means clustering. As suggested above, this is due to a difference in goals between LVQ and K-means clustering. K-means does not attempt to generate optimal decision lines in a categorization task; similarly, LVQ is not meant to minimize distortion. As our interest here is phoneme recognition accuracy, within the framework of the recognition system presented here, the LVQ algorithms are more appropriate than K-means clustering.

We saw above that with large numbers of reference vectors, LVQ and K-means have about the same performance. However, the fact that LVQ can achieve extremely high performance even with small numbers of reference vectors seems quite significant. The goal of vector quantization is to reduce data from a large number of training vectors to a small number of reference vectors. For the purposes of our phoneme recognition system, LVQ achieves this goal more fully. Furthermore, even for large numbers of reference vectors, LVQ performs slightly better than K-means.

LVQ2 seems clearly more powerful than LVQ1 in terms of ability to approximate optimal decision lines. Both algorithms use the same principle, of pushing incorrect vectors away and pulling correct vectors closer, but the manner in which this process is controlled in LVQ2 allows for much more careful adaptation of the decision boundaries in the task.

We have throughout mentioned that LVQ is a very fast algorithm, mainly due to the very good initial conditions obtained using K-means clustering. At the time of the TDNN implementation in [3], it took 4 days for TDNN to learn /b/, /d/, /g/. Thus the minute or so required for LVQ2, on the same computer, seemed to compare quite favorably. Recently, however, Haffner et al. [9] have achieved tremendous speed-ups for TDNN, such that we can no longer claim speed as a decisive advantage over TDNN. However, it still seems fair to say that LVQ, in its basic form, is faster than Back-propagation in its basic form.

We should mention that our LVQ networks are significantly bigger than their corresponding TDNNs, by a factor ranging from about 4 to about 13. Training speed is nonetheless very fast, and recognition of a single token is still done in better than real time. We should note that the architecture we present here is undoubtedly not the optimal LVQ architecture for phoneme recognition, and other architectures we are now considering may well allow for significantly smaller networks. Another point is that as a vector quantizer, LVQ is open to the many existing techniques for editing out unnecessary vectors, as well as for finding the closest vector in logarithmic time[2] .

We find the simplicity of the learning rule quite appealing. It involves no calculations of sigmoid functions or derivatives, the main calculation being that

[2] Although one concern is that the high dimensionality of the vectors used here may limit the effectiveness of these methods.

for finding the Euclidean distance between two vectors, which is very easy to implement and parallelize.

The results reported for other LVQ based architectures suggest that training on shifted speech segments improves performance, especially performance on test data, quite significantly. The shifting window scheme means that it is not essential that the center frame of the phoneme be perfectly aligned with the center frame of the input layer for it to be correctly recognized. This is quite an important property, as we cannot reasonably expect databases to be so accurately labelled. Another advantage of the shifting window scheme is that training on shifted segments in a way results in an expansion of the training data, which might be beneficial.

Finally, the results for an LVQ system with a different recognition rule provide some motivation for the main recognition rule presented here. Although very simple, this recognition rule seems to be able to integrate information over time so as to provide a more robust decision than that obtained by using only one reference vector to identify a whole token.

7. Conclusion

Our results indicate that LVQ can be the basis of a simple but very powerful shift-tolerant classifier for phoneme recognition. Our LVQ based system achieved recognition rates in the 98%-99% correct range for 7 Japanese phonemic classes, and a recognition rate of nearly 98% for all Japanese consonants. Our results compare favorably with those obtained in the TDNN system developed by Waibel et al.[3].

On the whole, LVQ seems to be endowed with great simplicity, speed, and powerful classification ability. Furthermore, since LVQ is a vector quantizer, it is open to the many techniques already developed for VQ systems in speech processing, such as speaker adaptation and Dynamic Time Warping. It could also be used in combination with Hidden Markov Models; one very simple possibility is simply to use LVQ to generate the HMM codebook. Thus, it seems that the LVQ system we present here could be extended to serve as the basis for a successful speech recognition system.

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] 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.

[4] A. Waibel, H. Sawai, K. Shikano, *Modularity and Scaling in Large Phonemic Neural Networks*, Technical Report TR-I-0034, ATR Interpreting Telephony Research Laboratories, August 5, 1988.

[5] E. McDermott, S. Katagiri, Phoneme Recognition Using Kohonen Networks, *Proc. of ATR Neural Net Workshop*, July 1988.

[6] E. McDermott, S. Katagiri, Shift-invariant Phoneme Recognition Using Kohonen Networks, *Proc. of Acoustical Society of Japan*, October 1988.

[7] J. Makhoul, S. Roucos, and H. Gish, (1985), Vector Quantization in Speech Coding, *Proc IEEE 73*, No. 11, 1551-88.

[8] M. Yokota, S. Katagiri and E. McDermott; Learning in an LVQ Based Phoneme Recognition System, *IEICE Technical Report*, SP88-104, pp. 65-72, December 1988

[9] P. Haffner, A. Waibel, K. Shikano; *Fast Back-Propagation Learning Methods for Neural Networks in Speech*, Technical Report TR-I-0058, ATR Interpreting Telephony Research Laboratories, November 1988.

[10] D.E. Rumelhart, Learning and Generalization: The Role of Minimal Networks, *Proc. of the ATR Workshop on Neural Networks and PDP*, July 1988, Osaka.

[11] Duda, R.O., Hart, P.E.: *Pattern Classification and Scene Analysis* , Chapter 2, Wiley, New York, 1973.