Phoneme Recognition Using Time-Delay Neural Networks

ALEXANDER WAIBEL, MEMBER, IEEE, TOSHIYUKI HANAZAWA, GEOFFREY HINTON, KIYOHISO SHIKANO, MEMBER, IEEE, AND KEVIN J. LANG

Abstract—In this paper we present a Time-Delay Neural Network (TDNN) approach to phoneme recognition which is characterized by two important properties. 1) Using a 3 layer arrangement of simple computing units, a hierarchy can be constructed that allows for the formation of arbitrary nonlinear decision surfaces. The TDNN learns these decision surfaces automatically using error backpropagation [1]. 2) The time-delay arrangement enables the network to discover acoustic–phonetic features and the temporal relationships between them independent of position in time and hence not blurred by temporal shifts in the input.

As a recognition task, the speaker-dependent recognition of the phonemes “B,” “D,” and “G” in varying phonetic contexts was chosen. For comparison, several discrete Hidden Markov Models (HMM) were trained to perform the same task. Performance evaluation over 1946 testing tokens from three speakers showed that the TDNN achieves a recognition rate of 98.5 percent correct while the rate obtained by the best of our HMM’s was only 93.7 percent. Closer inspection reveals that the network “invented” well-known acoustic–phonetic features (e.g., F2-rise, F2-fall, vowel-onset) as useful abstractions. It also developed alternate internal representations to link different acoustic realizations to the same concept.

I. INTRODUCTION

In recent years, the advent of new learning procedures and the availability of high speed parallel supercomputers have given rise to a renewed interest in connectionist models of intelligence [1]. Sometimes also referred to as artificial neural networks or parallel distributed processing models, these models are particularly interesting for cognitive tasks that require massive constraint satisfaction, i.e., the parallel evaluation of many clues and facts and their interpretation in the light of numerous interrelated constraints. Cognitive tasks, such as vision, speech, language processing, and motor control, are also characterized by a high degree of uncertainty and variability and it has proved difficult to achieve good performance for these tasks using standard serial programming methods. Complex networks composed of simple computing units are attractive for these tasks not only because of their “brain-like” appeal but because they offer ways for automatically designing systems that can make use of multiple interacting constraints. In general, such constraints are too complex to be easily programmed and require the use of automatic learning strategies. Such learning algorithms now exist (for an excellent review, see Lippman [2]) and have been demonstrated to discover interesting internal abstractions in their attempts to solve a given problem [1], [3]-[5]. Learning is most effective, however, when used in an architecture that is appropriate for the task. Indeed, applying one’s prior knowledge of a task domain and its properties to the design of a suitable neural network model might well prove to be a key element in the successful development of connectionist systems.

Naturally, these techniques will have far-reaching implications for the design of automatic speech recognition systems, if proven successful in comparison to already-existing techniques. Lippmann [6] has compared several kinds of neural networks to other classifiers and evaluated their ability to create complex decision surfaces. Other studies have investigated actual speech recognition tasks and compared them to psychological evidence in speech perception [7] or to existing speech recognition techniques [8], [9]. Speech recognition experiments using neural nets have so far mostly been aimed at isolated word recognition (mostly the digit recognition task) [10]-[13] or phonetic recognition with predefined constant [14], [15] or variable phonetic contexts [16], [14], [17].

A number of these studies report very encouraging recognition performance [16], but only few comparisons to existing recognition methods exist. Some of these comparisons found performance similar to existing methods [9], [11], but others found that networks perform worse than other techniques [8]. One might argue that this state of affairs is encouraging considering the amount of fine-tuning that has gone into optimizing the more popular, established techniques. Nevertheless, better comparative performance figures are needed before neural networks can be considered as a viable alternative for speech recognition systems.

1The uninitiated reader should be cautioned not to overinterpret the now-popular term “neural network.” Although these networks appear to mimic certain properties of neural cells, no claim can be made that present exploratory attempts simulate the complexities of the human brain.
One possible explanation for the mixed performance results obtained so far may be limitations in computing resources leading to shortcuts that limit performance. Another more serious limitation, however, is the inability of most neural network architectures to deal properly with the dynamic nature of speech. Two important aspects of this are for a network to represent temporal relationships between acoustic events, while at the same time providing for invariance under translation in time. The specific movement of a formant in time, for example, is an important cue to determining the identity of a voiced stop, but it is irrelevant whether the same set of events occurs a little sooner or later in the course of time. Without translation invariance, a neural net requires precise segmentation to align the input pattern properly. Since this is not always possible in practice, learned features tend to get blurred (in order to accommodate slight misalignments) and their performance deteriorates. In general, shift invariance has been recognized as a critically important property for connectionist systems and a number of promising models have been proposed for speech and other domains [18]-[22], [14], [17], [22].

In the present paper, we describe a Time-Delay Neural Network (TDNN) which addresses both of these aspects of speech and demonstrate through extensive performance evaluation that superior recognition results can be achieved using this approach. In the following section, we begin by introducing the architecture and learning strategy of a TDNN aimed at phoneme recognition. Next, we compare the performance of our TDNN's to one of the more popular current recognition techniques: Hidden Markov Models (HMM). In Section III, we start by describing an HMM, under development at ATR [23], [24]. Both techniques, the TDNN and the HMM, are then evaluated over a testing database and we report the results. We show that substantially higher recognition performance is achieved by the TDNN than by the best of our HMM's. In Section IV, we then take a closer look at the internal representation that the TDNN learns for this task. It discovers a number of interesting linguistic abstractions which we show by way of examples. The implications of these results are then discussed and summarized in the final section of this paper.

II. TIME-DELAY NEURAL NETWORKS (TDNN)

To be useful for speech recognition, a layered feedforward neural network must have a number of properties. First, it should have multiple layers and complex interconnections between units in each of these layers. This is to ensure that the network will have the ability to learn complex nonlinear decision surfaces [2], [6]. Second, the network should have the ability to represent relationships between events in time. These events could be spectral coefficients, but might also be the output of higher level feature detectors. Third, the actual features or abstractions learned by the network should be invariant under translation in time. Fourth, the learning procedure should not require precise temporal alignment of the labels that are to be learned. Fifth, the number of weights in the network should be sufficiently small compared to the amount of training data so that the network is forced to encode the training data by extracting regularity. In the following, we describe a TDNN architecture that satisfies all of these criteria and is designed explicitly for the recognition of phonemes, in particular, the voiced stops "B," "D," and "G."

A. A TDNN Architecture for Phoneme Recognition

The basic unit used in many neural networks computes the weighted sum of its inputs and then passes this sum through a nonlinear function, most commonly a threshold or sigmoid function [2], [1]. In our TDNN, this basic unit is modified by introducing delays $D_i$ through $D_j$ as shown in Fig. 1. The $J$ inputs of such a unit now will be multiplied by several weights, one for each delay and one for the undelayed input. For $J = 2$, and $J = 16$, for example, 48 weights will be needed to compute the weighted sum of the 16 inputs, with each input now measured at three different points in time. In this way, a TDNN unit has the ability to relate and compare current input to the past history of events. The sigmoid function was chosen as the nonlinear output function $F$ due to its convenient mathematical properties [18], [5].

For the recognition of phonemes, a three layer net is constructed. It's overall architecture and a typical set of activities in the units are shown in Fig. 2.

At the lowest level, 16 normalized mel-scale spectral coefficients serve as input to the network. Input speech, sampled at 12 kHz, was Hamming windowed and a 256-point FFT computed every 5 ms. Mel-scale coefficients were computed from the power spectrum by computing log energies in each mel-scale energy band [25], where adjacent coefficients in frequency overlap by one spectral sample and are smoothed by reducing the shared sample by 50 percent [25]. Adjacent coefficients in time were collapsed for further data reduction resulting in an overall 10 ms frame rate. All coefficients of an input token (in this case, 15 frames of speech centered around the hand-labeled vowel onset) were then normalized. This was accomplished by subtracting from each coefficient the average coefficient energy computed over all 15 frames of an input token and then normalizing each coefficient to lie between $-1$ and $+1$. All tokens in our database were preprocessed in the same fashion. Fig. 2 shows the resulting coefficients for the speech token "BA" as input to the

\[^{2}\text{Lippmann [2], [6] demonstrated recently that three layers can encode arbitrary pattern recognition decision surfaces. We believe that complex nonlinear decision surfaces are necessary to properly perform classification in the light of considerable acoustic variability as reported in the experiments below.}

\[^{3}\text{Naturally, a number of alternative signal representations could be used as input; but have not been tried in this study. Filterbank coefficients were chosen as they are simple to compute and readily-interpretable in the light of acoustic-phonetics. The mel-scale is a physiologically motivated frequency scale that provides better relative frequency resolution for lower frequency bands. Our implementation resulted in coefficients with a bandwidth of approximately 190 Hz up to 1400 Hz, and with increasing bandwidth thereafter.}\]
network, where positive values are shown as black squares and negative values as gray squares.

This input layer is then fully interconnected to a layer of 8 time-delay hidden units, where \( J = 16 \) and \( N = 2 \) (i.e., 16 coefficients over 3 frames with time delay 0, 1, and 2). An alternative way of viewing this is depicted in Fig. 2. It shows the inputs to these time-delay units expanded out spatially into a 3 frame window, which is passed over the input spectrogram. Each unit in the first hidden layer now receives input (via 48 weighted connections) from the coefficients in the 3 frame window. The particular choice of 3 frames (30 ms) was motivated by earlier studies [26]-[29] that suggest that a 30 ms window might be sufficient to represent low level acoustic-phonetic events for stop consonant recognition. It was also the optimal choice among a number of alternative designs evaluated by Lang [21] on a similar task.

In the second hidden layer, each of 3 TDNN units looks at a 5 frame window of activity levels in hidden layer 1 (i.e., \( J = 8, N = 4 \)). The choice of a larger 5 frame window in this layer was motivated by the intuition that higher level units should learn to make decisions over a wider range in time based on more local abstractions at lower levels.

Finally, the output is obtained by integrating (summing) the evidence from each of the 3 units in hidden layer 2 over time and connecting it to its pertinent output unit (shown in Fig. 2 over 9 frames for the "B" output unit). In practice, this summation is implemented simply as another nonlinear (sigmoid function is applied here as well) TDNN unit which has fixed equal weights to a row of unit firings over time in hidden layer 2.

When the TDNN has learned its internal representation, it performs recognition by passing input speech over the TDNN units. In terms of the illustration of Fig. 2, this is equivalent to passing the time-delay windows over the lower level units' firing patterns. At the lowest level, these firing patterns simply consist of the sensory input, i.e., the spectral coefficients.

Each TDNN unit outlined in this section has the ability to encode temporal relationships within the range of the \( N \) delays. Higher layers can attend to larger time spans, so local short duration features will be formed at the lower layer and more complex longer duration features at the higher layer. The learning procedure ensures that each of the units in each layer has its weights adjusted in a way that improves the network's overall performance.

### B. Learning in a TDNN

Several learning techniques exist for optimization of neural networks [1], [2], [30]. For the present network, we adopt the Backpropagation Learning Procedure [18],

\[ \text{Note, however, that as for all units in this network (except the input units), the output units are also connected to a permanently active threshold unit. In this way, both an output unit's one shared connection to a row in hidden layer 2 and its dc-bias are learned and can be adjusted for optimal classification.} \]

\[ \text{Thus, 13 frames of activations in hidden layer 1 are generated when scanning the 15 frames of input speech with a 3 frame time delay window. Similarly, 9 frames are produced in hidden layer 2 from the 13 frames of activation in the layer below.} \]
Mathematically, backpropagation is gradient descent of the mean-squared error as a function of the weights. The procedure performs two passes through the network. During the forward pass, an input pattern is applied to the network with its current connection strengths (initially small random weights). The outputs of all the units at each level are computed starting at the input layer and working forward to the output layer. The output is then compared to the desired output and its error calculated. During the backward pass, the derivative of this error is then propagated back through the network, and all the weights are adjusted so as to decrease the error [18], [5]. This is repeated many times for all the training tokens until the network converges to producing the desired output.

In the previous section, we described a method of expressing temporal structure in a TDNN and contrasted this method to training a network on a static input pattern (spectrogram), which results in shift sensitive networks (i.e., poor performance for slightly misaligned input patterns) as well as less crisp decision making in the units of the network (caused by misaligned tokens during training).

To achieve the desired learning behavior, we need to ensure that the network is exposed to sequences of patterns and that it is allowed (or encouraged) to learn about the most powerful cues and sequences of cues among them. Conceptually, the backpropagation procedure is applied to speech patterns that are stepped through in time. An equivalent way of achieving this result is to use a spatially expanded input pattern, i.e., a spectrogram plus some constraints on the weights. Each collection of TDNN units described above is duplicated for each time frame shifted in time. In this way, the whole history of activities is available at once. Since the shifted copies of the TDNN units are mere duplicates and are to look for the same acoustic event, the weights of the corresponding connections in the time shifted copies must be constrained to be the same. To implement this, we first apply the regular backpropagation forward and backward pass to all time-shifted copies as if they were separate events. This yields different error derivatives for corresponding (time shifted) connections. Rather than changing the weights on time-shifted connections separately, however, we actually update each weight on corresponding connections by the same value, namely by the average of all corresponding time-delayed weight changes. Fig. 2 illustrates this by showing in each layer only two connections that are linked to (constrained to have the same value as) their time-shifted neighbors. Of course, this applies to all connections and all time shifts. In this way, the network is forced to discover useful acoustic-phonetic features in the input, regardless of when in time they actually occurred. This is an important property, as it makes the network independent of error-prone preprocessing algorithms that otherwise would be needed for time alignment and/or segmentation. In Section IV-C, we will show examples of grossly misaligned patterns that are properly recognized due to this property.

The procedure described here is computationally rather expensive, due to the many iterations necessary for learning a complex multidimensional weight space and the number of learning samples. In our case, about 800 learning samples were used, and between 20,000 and 50,000 iterations of the backpropagation loop were run over all training samples. Two steps were taken to perform learning within reasonable time. First, we have implemented our learning procedure in C and Fortran on a 4 processor Alliant supercomputer. The speed of learning can be improved considerably by computing the forward and backward sweeps for several different training samples in parallel on different processors. Further improvements can be gained by vectorizing operations and possibly assembling coding the innermost loop. Our present implementation achieves about a factor of 9 speedup over a VAX 8600, but still leaves room for further improvements (Lang [21], for example, reports a speedup of a factor of 120 over a VAX11/780 for an implementation running on a Convex supercomputer). The second step taken toward improving learning time is given by a staged learning strategy. In this approach, we start optimizing the network based on 3 prototypical training tokens only. In this case, convergence is achieved rapidly, but the network will have learned a representation that generalizes poorly to new and different patterns. Once convergence is achieved, the network is presented with approximately twice the number of tokens and learning continues until convergence of the whole history of activities is achieved.

Fig. 3 shows the progress during a typical learning run. The measured error is 1/2 the squared error of all the output units, normalized for the number of training tokens. In this run, the number of training tokens used were 3, 6, 9, 24, 99, 249, and 780. As can be seen from Fig. 3, the error briefly jumps up every time more variability is introduced by way of more training data. The network is then forced to improve its representation to discover clues that generalize better and to deemphasize those that turn out to be merely irrelevant idiosyncracies of a limited sample set. Using the full training set of 780 tokens, this particular run was continued until iteration 35,000 (Fig. 3 shows the learning curve only up to 15,000 iterations). With this full training set, small learning steps have to be taken and learning progresses slowly. In this case, a step size of 0.002 and a momentum of 0.1 was used. The staged learning approach was found to be useful to move the weights of the network rapidly into the neighborhood of a reasonable solution, before the rather slow fine tuning over all training tokens begins.

Despite these speedups, learning runs still take in the
order of several days. A number of programming tricks [21] as well as modifications to the learning procedure
[31] are not implemented yet and could yield another factor of 10 or more in learning time reduction. It is important to
note, however, that the amount of computation considered here is necessary only for learning of a TDNN and not for
recognition. Recognition can easily be performed in better than real time on a workstation or personal computer. The
simple structure makes TDNN’s also well suited for standardized VLSI implementation. The detailed knowledge could be
learned “off-line” using substantial computing power and then downloaded in the form of weights onto a real-time
production network.

III. RECOGNITION EXPERIMENTS

We now turn to an experimental evaluation of the TDNN’s recognition performance. In particular, we would like to compare the TDNN’s performance to the performance of the currently most popular recognition
method: Hidden Markov Models (HMM). For the performance evaluation reported here, we have chosen the best of
a number of HMM’s developed in our laboratory. Several other HMM-based variations and models have been
tried in an effort to optimize our HMM, but we make no claim that an exhaustive evaluation of all HMM-based

techniques was accomplished. We should also point out that the experiments reported here were aimed at evalu-
ing two different recognition philosophies. Each recognition method was therefore implemented and optimized
using its preferred representation of the speech signal, i.e., a representation that is well suited and most commonly
used for the method evaluated. Evaluation of both methods was of course carried out using the same speech
input data, but we caution the reader that due to the differences in representation, the exact contribution to over-
all performance of the recognition strategy as opposed to its signal representation is not known. It is conceivable
that improved front end processing might lead to further performance improvements for each technique. In the
following sections, we will start by introducing the best of our Hidden Markov Models. We then describe the
experimental conditions and the database used for performance evaluation and conclude with the performance results
achieved by our TDNN and HMM.

A. A Hidden Markov Model (HMM) for Phoneme Recognition

HMM’s are currently the most successful and promising approach [32]–[34] in speech recognition as they have
been successfully applied to the whole range of recognition tasks. Excellent performance was achieved at all
levels from the phonemic level [35]–[38] to word recognition [39], [34] and to continuous speech recognition [40]. The
success of HMM’s is partially due to their ability to cope with the variability in speech by means of stochastic
modeling. In this section, we describe an HMM developed in our laboratory that was aimed at phoneme recognition,
more specifically the voiced stops ‘‘B,’’ ‘‘D,’’ and ‘‘G.’’ The model described was the best of a number of alternate
designs developed in our laboratory [23], [24].

The acoustic front end for Hidden Markov Modeling is typically a vector quantizer that classifies sequences of
short-time spectra. Such a representation was chosen as it is highly effective for HMM-based recognizers [40].

Input speech was sampled at 12 kHz, preemphasized by

\[1 - 0.97 z^{-1}\]

and windowed using a 256-point Ham-
ming window every 3 ms. Then a 12-order LPC analysis was
carried out. A codebook of 256 LPC spectrum en-
velopes was generated from 216 phonetically balanced
words. The Weighted Likelihood Ratio [41], [42] aug-
mented with power values (PWLR) [43], [42] was used as
LPC distance measure for vector quantization.

A fairly standard HMM was adopted in this paper as
shown in Fig. 4. It has four states and six transitions and
was found to be the best of a series of alternate models
tried in our laboratory. These included models with two,
three, four, and five states and with tied arcs and null arcs
[23], [24].

The HMM probability values were trained using vector
sequences of phonemes according to the forward–back-
ward algorithm [32]. The vector sequences for ‘‘B,’’
‘‘D,’’ and ‘‘G’’ include a consonant part and five frames
of the following vowel. This is to model important tran-
sient information, such as formant movement, and has
lead to improvements over context insensitive models
[23], [24]. Again, variations on these parameters have
been tried for the discrimination of these three voiced stop
consonants. In particular, we have used 10 and 15 frames
(i.e., 30 and 45 ms) of the following vowel in a 5 state
HMM, but no performance improvements over the model
described were obtained.

The HMM was trained using about 250 phoneme to-
kens of vector sequences per speaker and phoneme (see
details of the training database below). Fig. 5 shows for
a typical training run the average log probability normal-
ized by the number of frames. Training was continued
until the increase of the average log probability between iterations became less than $2 \times 10^{-4}$.

Typically, about 10–20 learning iterations are required for 256 tokens. A training run takes about 1 h on a VAX 8700. Floor values\(^8\) were set on the output probabilities to avoid errors caused by zero probabilities. We have experimented with composite models, which were trained using a combination of context-independent and context-dependent probability values as suggested by Schwartz et al. [35], [36]. In our case, no significant improvements were attained.

**B. Experimental Conditions**

For performance evaluation, we have used a large vocabulary database of 5240 common Japanese words [44]. These words were uttered in isolation by three male native Japanese speakers (MAU, MHT, and MNM, all professional announcers) in the order they appear in a Japanese dictionary. All utterances were recorded in a sound-proof booth and digitized at a 12 kHz sampling rate. The database was then split into a training set (the even numbered files as derived from the recording order) and a testing set (the odd numbered files). A given speaker’s training and testing data, therefore, consisted of 2620 utterances each, from which the actual phonetic tokens were extracted.

The phoneme recognition task chosen for this experiment was the recognition of the voiced stops, i.e., the phonemes “B,” “D,” and “G.” The actual tokens were extracted from the utterances using manually selected acoustic–phonetic labels provided with the database [44]. For speaker MAU, for example, a total of 219 “B’s,” 203 “D’s,” and 260 “G’s” were extracted from the training and 227 “B’s,” 179 “D’s,” and 252 “G’s,” from the testing data. Both recognition schemes, the TDNN’s and the HMM’s, were trained and tested speaker dependently. Thus, in both cases, separate networks were trained for each speaker.

In our database, no preselection of tokens was performed. All tokens labeled as one of the three voiced stops were included. It is important to note that since the consonant tokens were extracted from entire utterances and not read in isolation, a significant amount of phonetic variability exists. Foremost, there is the variability introduced by the phonetic context out of which a token is extracted. The actual signal of a “BA” will therefore look significantly different from a “BI” and so on. Second, the position of a phonemic token within the utterance introduces additional variability. In Japanese, for example, a “G” is nasalized, when it occurs embedded in an utterance, but not in utterance initial position. Both of our recognition algorithms are only given the phonemic identity of a token and must find their own ways of representing the fine variations of speech.

**C. Results**

Table I shows the results from the recognition experiments described above as obtained from the testing data. As can be seen, for all three speakers, the TDNN yields considerably higher performance than our HMM. Averaged over all three speakers, the error rate is reduced from 6.3 to 1.5 percent—a more than fourfold reduction in error.

While it is particularly important here to base performance evaluation on testing data,\(^9\) a few observations can be made from recognition runs over the training data. For the training data set, recognition error rates were: 99.6 percent (MAU), 99.7 percent (MHT), and 99.7 percent (MNM) for the TDNN and 96.9 percent (MAU), 99.1 percent (MHT), and 95.7 percent (MNM) for the HMM. Comparison of these results to those from the testing data in Table I indicates that both methods achieved good generalization from the training set to unknown data. The data also suggest that better classification rather than better generalization might be the cause of the TDNN’s better performance shown in Table I.

Figs. 6–11 show scatter plots of the recognition outcome for the test data for speaker MAU, using the HMM and the TDNN. For the HMM (see Figs. 6–8), the log probability of the next best matching incorrect token is plotted against the log probability of the correct token, e.g., “B,” “D,” and “G.” In Figs. 9–11, the activation levels from the TDNN’s output units are plotted in the same fashion. Note that these plots are not easily comparable, as the two recognition methods have been trained in quite different ways. They do, however, represent the

\(^9\)If the training data are insufficient, neural networks can in principle learn to memorize training patterns rather than finding generalization of speech.

\(^{10}\)Normalized by number of frames.
TABLE 1
RECOGNITION RESULTS FOR THREE SPEAKERS OVER TEST DATA USING TDNN AND HMM

<table>
<thead>
<tr>
<th>speaker</th>
<th>number of tokens</th>
<th>number of errors</th>
<th>recognition rate</th>
<th>number of errors</th>
<th>recognition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAU</td>
<td>1375</td>
<td>3</td>
<td>98.8</td>
<td>18</td>
<td>92.1</td>
</tr>
<tr>
<td>MHT</td>
<td>200</td>
<td>2</td>
<td>99.9</td>
<td>8</td>
<td>96.2</td>
</tr>
<tr>
<td>MNN</td>
<td>16</td>
<td>11</td>
<td>97.5</td>
<td>27</td>
<td>87.6</td>
</tr>
</tbody>
</table>

Numerical values that each method's decision rule uses to determine the recognition outcome. We present these plots here to show some interesting properties of the two techniques. The most striking observation that can be made from these plots is that the output units of a TDNN have a tendency to fire with high confidence as can be seen from the plots.
from the cluster of dots in the lower right-hand corner of the scatter plots. Most output units tend to fire strongly for the correct phonemic class and not at all for any other, a property that is encouraged by the learning procedure. One possible consequence of this is that rejection thresholds could be introduced to improve recognition performance. If one were to eliminate among speaker MAU's tokens all those whose highest activation level is less than 0.5 and those which result in two or more closely competing activations (i.e., are near the diagonal in the scatter plots), 2.6 percent of all tokens would be rejected, while the remaining substitution error rate would be less than 0.46 percent.

IV. THE LEARNED INTERNAL REPRESENTATIONS OF A TDNN

Given the encouraging performance of our TDNN's, a closer look at the learned internal representation of the network is warranted. What are the properties or abstractions that the network has learned that appear to yield a very powerful description of voiced stops? Figs. 12 and 13 show two typical instances of a "D" out of two different phonetic contexts ("DA" and "DO," respectively). In both cases, only the correct unit, the "D-output unit," fires strongly, despite the fact that the two input spectrograms differ considerably from each other. If we study the internal firings in these two cases, we can see that the network has learned to use alternate internal representations to link variations in the sensory input to the same higher level concepts. A good example is given by the firings of the third and fourth hidden unit in the first layer above the input layer. As can be seen from Fig. 13, the fourth hidden unit fires particularly strongly after vowel onset in the case of "DO," while the third unit shows stronger activation after vowel onset in the case of "DA."

Fig. 14 shows the significance of these different firing patterns. Here the connection strengths for the eight moving TDNN units are shown, where white and black blobs represent positive and negative weights, respectively, and the magnitude of a weight is indicated by the size of the
blob. In this figure, the time delays are displayed spatially as a 3 frame window of 16 spectral coefficients. Conceptually, the weights in this window form a moving acoustic-phonetic feature detector that fires when the pattern for which it is specialized is encountered in the input speech. In our example, we can see that hidden unit number 4 (which was activated for "DO") has learned to fire when a rising (or falling) second formant starting at around 1600 Hz is found in the input (see filled arrow in Fig. 14). As can be seen in Fig. 13, this is the case for "DO" and hence the firing of hidden unit 4 after voicing onset (see row pointed to by the filled arrow in Fig. 13). In the case of "DA" (see Fig. 12), in turn, the second formant does not fall significantly, and hidden unit 3 (pointed to by the filled arrow) fires instead. From Fig. 14 we can verify that TDNN unit 3 has learned to look for a steady (or only slightly falling) second formant starting at about 1800 Hz. The connections in the second and third layer then link the different firing patterns observed in the first hidden layer into one and the same decision.

Another interesting feature can be seen in the bottom hidden unit in hidden layer number 1 (see Figs. 12 and 13, and compare them to the weights of hidden unit 1 displayed in Fig. 14). This unit has learned to take on the role of finding the segment boundary of the voiced stop. It does so in reverse polarity, i.e., it is always on except when the vowel onset of the voiced stop is encountered (see unfilled arrow in Figs. 13 and 12). Indeed, the higher layer TDNN units subsequently use this "segmenter" to base the final decision on the occurrence of the right lower features at the right point in time.

In the previous example, we have seen that the TDNN can account for variations in phonetic context. Figs. 15 and 16 show examples of variability caused by the relative position of a phoneme within a word. In Japanese, a "G" embedded in a word tends to be nasalized as seen in the spectrum of a "GA" in Fig. 15. Fig. 16 shows a word initial "GA." Despite the striking differences between these two input spectrograms, the network's internal alternate representations manage to produce in both cases crisp output firings for the right category.

Figs. 17 and 18, finally, demonstrate the shift invariance of the network. They show the same token "DO" of Fig. 13, misaligned by +30 ms and -30 ms, respectively. Despite the gross misalignment (note that significant transitional information is lost by the misalignment in Fig. 18), the correct result was obtained reliably. A close look at the internal activation patterns reveals that the hidden units' feature detectors do indeed fire according to the events in the input speech, and are not negatively affected by the relative shift with respect to the input units. Naturally, error rates will gradually increase when the tokens are artificially shifted to an extent that important features begin to fall outside the 15 frame data window considered here. We have observed, for example, a 2.6 percent increase in error rate when all tokens from

---

![Fig. 15. TDNN activation patterns for "GA" embedded in an utterance.](image-url)
the training data were artificially shifted by 20 ms. Such residual time-shift sensitivities are due to the edge effects at the token boundaries and can probably be removed by training the TDNN using randomly shifted training tokens. We also consider the formation of shift-invariant internal features to be the important desirable property we observe in the TDNN. Such internal features could be incorporated into larger speech recognition systems using more sophisticated search techniques or a syllable or word level TDNN, and hence could replace the simple integration layer we have used here for training and evaluation.

Three important properties of the TDNN's have been observed. First, our TDNN was able to learn, without human interference, meaningful linguistic abstractions such as formant tracking and segmentation. Second, we have demonstrated that it has learned to form alternate representations linking different acoustic events with the same higher level concept. In this fashion, it can implement trading relations between lower level acoustic events leading to robust recognition performance. Third, we have seen that the network is shift invariant and does not rely on precise alignment or segmentation of the input.

V. CONCLUSION AND SUMMARY

In this paper we have presented a Time-Delay Neural Network (TDNN) approach to phoneme recognition. We gratefully acknowledge one of the reviewers for suggesting this idea.