7. Classification Networks

Neural networks can be taught to map an input space to any kind of output space. For example, in the previous chapter we explored a homomorphic mapping, in which the input and output space were the same, and the networks were taught to make predictions or interpolations in that space.

Another useful type of mapping is *classification*, in which input vectors are mapped into one of *N* classes. A neural network can represent these classes by *N* output units, of which the one corresponding to the input vector's class has a "1" activation while all other outputs have a "0" activation. A typical use of this in speech recognition is mapping speech frames to phoneme classes. Classification networks are attractive for several reasons:

- They are simple and intuitive, hence they are commonly used.
- They are naturally discriminative.
- They are modular in design, so they can be easily combined into larger systems.
- They are mathematically well-understood.
- They have a probabilistic interpretation, so they can be easily integrated with statistical techniques like HMMs.

In this chapter we will give an overview of classification networks, present some theory about such networks, and then describe an extensive set of experiments in which we optimized our classification networks for speech recognition.

7.1. Overview

There are many ways to design a classification network for speech recognition. Designs vary along five primary dimensions: network architecture, input representation, speech models, training procedure, and testing procedure. In each of these dimensions, there are many issues to consider. For instance:

Network architecture (see Figure 7.1). How many layers should the network have, and how many units should be in each layer? How many time delays should the network have, and how should they be arranged? What kind of transfer function should be used in each layer? To what extent should weights be shared? Should some of the weights be held to fixed values? Should output units be integrated over time? How much speech should the network see at once?

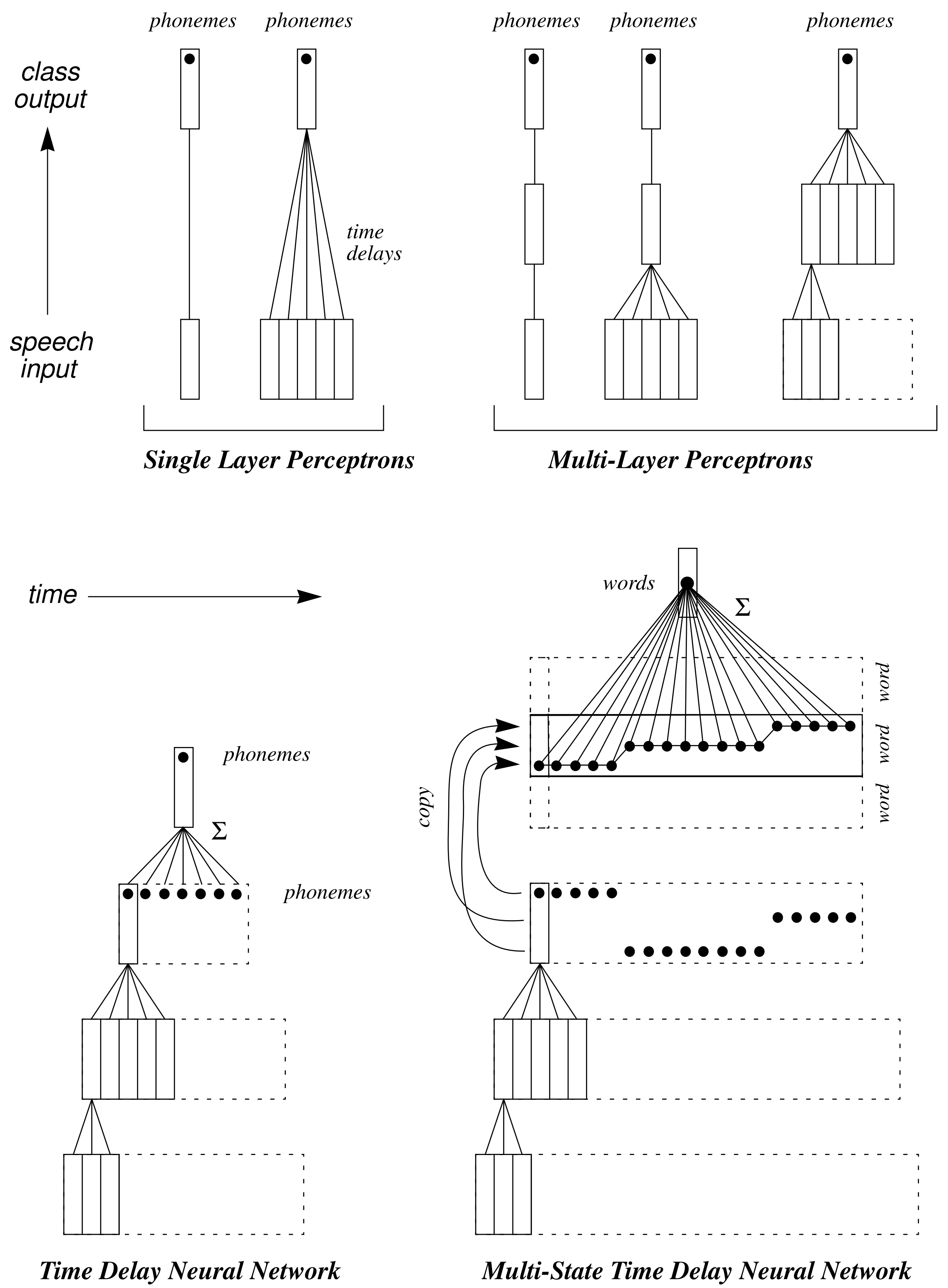


Figure 7.1: Types of network architectures for classification.

Input representation. What type of signal processing should be used? Should the resulting coefficients be augmented by redundant information (deltas, etc.)? How many input coefficients should be used? How should the inputs be normalized? Should LDA be applied to enhance the input representation?

Speech models. What unit of speech should be used (phonemes, triphones, etc.)? How many of them should be used? How should context dependence be implemented? What is the optimal phoneme topology (states and transitions)? To what extent should states be shared? What diversity of pronunciations should be allowed for each word? Should function words be treated differently than content words?

Training procedure. At what level (frame, phoneme, word) should the network be trained? How much bootstrapping is necessary? What error criterion should be used? What is the best learning rate schedule to use? How useful are heuristics, such as momentum or derivative offset? How should the biases be initialized? Should the training samples be randomized? Should training continue on samples that have already been learned? How often should the weights be updated? At what granularity should discrimination be applied? What is the best way to balance positive and negative training?

Testing procedure. If the Viterbi algorithm is used for testing, what values should it operate on? Should it use the network's output activations directly? Should logarithms be applied first? Should priors be factored out? If training was performed at the word level, should word level outputs be used during testing? How should duration constraints be implemented? How should the language model be factored in?

All of these questions must be answered in order to optimize a NN-HMM hybrid system for speech recognition. In this chapter we will try to answer many of these questions, based on both theoretical arguments and experimental results.

7.2. Theory

7.2.1. The MLP as a Posterior Estimator

It was recently discovered that if a multilayer perceptron is asymptotically trained as a 1-of-N classifier using mean squared error (MSE) or any similar criterion, then its output activations will approximate the posterior class probability $P(class|input)$, with an accuracy that improves with the size of the training set. This important fact has been proven by Gish (1990), Bourlard & Wellekens (1990), Hampshire & Pearlmutter (1990), Ney (1991), and others; see Appendix B for details.

This theoretical result is empirically confirmed in Figure 7.2. A classifier network was trained on a million frames of speech, using softmax outputs and cross entropy training, and then its output activations were examined to see how often each particular activation value was associated with the correct class. That is, if the network's input is x, and the network's kth output activation is $y_k(x)$, where k=c represents the correct class, then we empirically measured $P(k{=}c|y_k(x))$, or equivalently $P(k{=}c|x)$, since $y_k(x)$ is a direct function of x in the trained network. In the graph, the horizontal axis shows the activations $y_k(x)$, and the vertical axis shows the empirical values of $P(k{=}c|x)$. (The graph contains ten bins, each with about 100,000 data points.) The fact that the empirical curve nearly follow a 45 degree angle indicates that the network activations are indeed a close approximation for the posterior class probabilities.

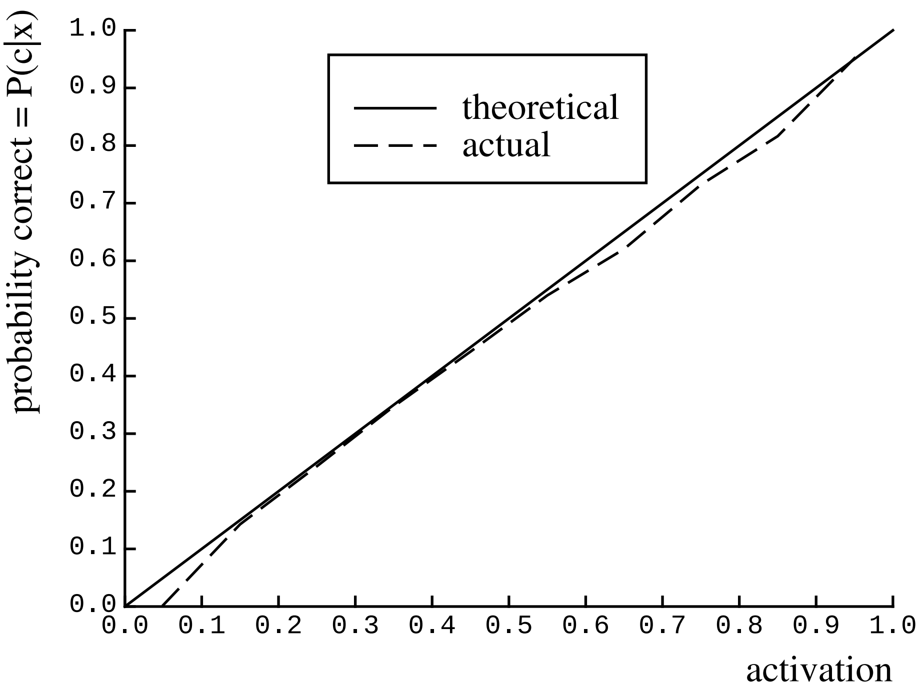


Figure 7.2: Network output activations are reliable estimates of posterior class probabilities.

Many speech recognition systems have been based on DTW applied directly to network class output activations, scoring hypotheses by summing the activations along the best alignment path. This practice is suboptimal for two reasons:

- The output activations represent probabilities, therefore they should be multiplied rather than added (alternatively, their logarithms may be summed).
- In an HMM, emission probabilities are defined as likelihoods $P(x|c)$, not as posteriors $P(c|x)$; therefore, in a NN-HMM hybrid, during recognition, the posteriors should first be converted to likelihoods using Bayes Rule:

$$P(x|c) = \frac{P(c|x) \cdot P(x)}{P(c)} \tag{72}$$

where $P(x)$ can be ignored during recognition because it's a constant for all states in any given frame, so the posteriors $P(c|x)$ may be simply divided by the priors $P(c)$. Intuitively, it can be argued that the priors should be factored out because they are already reflected in the language model (grammar) used during testing.

Bourlard and Morgan (1990) were the first to demonstrate that word accuracy in a NN-HMM hybrid can be improved by using $\log(y/P(c))$ rather than the output activation y itself in Viterbi search. We will provide further substantiation of this later in this chapter.

7.2.2. Likelihoods vs. Posteriors

The difference between likelihoods and posteriors is illustrated in Figure 7.3. Suppose we have two classes, c_1 and c_2. The likelihood $P(x|c_i)$ describes the distribution of the input x given the class, while the posterior $P(c_i|x)$ describes the probability of each class c_i given the input. In other words, likelihoods are independent density models, while posteriors indicate how a given class distribution compares to all the others. For likelihoods we have $\int_x P(x|c_i) = 1$, while for posteriors we have $\sum_i P(c_i|x) = 1$.

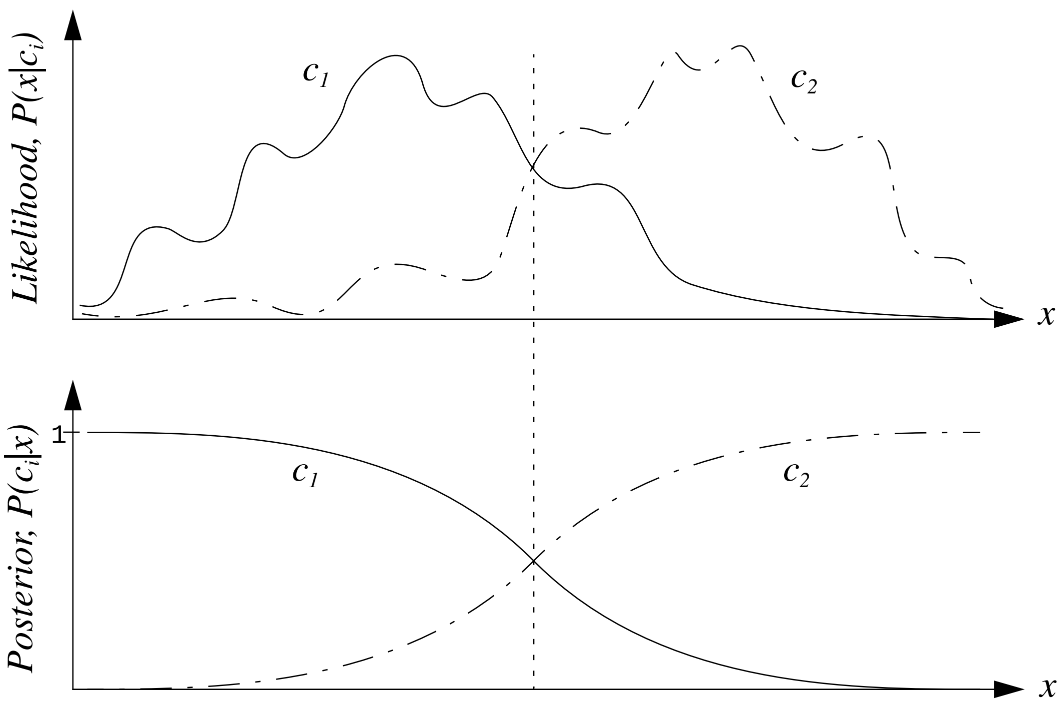


Figure 7.3: Likelihoods model independent densities; posteriors model their comparative probability.

Posteriors are better suited to classifying the input: the Bayes decision rule tells us that we should classify x into class c_1 iff

$$P(c_1|x) > P(c_2|x) .$$

If we wanted to classify the input using likelihoods, we would first have to convert these posteriors into likelihoods using Bayes Rule, yielding a more complex form of the Bayes decision rule which says that says we should classify x into class c_1 iff

$$P(x|c_1)\,P(c_1) > P(x|c_2)\,P(c_2) \tag{73}$$

Note that the priors $P(c_i)$ are implicit in the posteriors, but not in likelihoods, so they must be explicitly introduced into the decision rule if we are using likelihoods.

Intuitively, likelihoods model the surfaces of distributions, while posteriors model the boundaries between distributions. For example, in Figure 7.3, the bumpiness of the distributions is modeled by the likelihoods, but the bumpy surface is ignored by the posteriors, since the boundary between the classes is clear regardless of the bumps. Thus, likelihood models (as used in the states of an HMM) may have to waste their parameters modeling irrelevant details, while posterior models (as provided by a neural network) can represent critical information more economically.

7.3. Frame Level Training

Most of our experiments with classification networks were performed using frame level training. In this section we will describe these experiments, reporting the results we obtained with different network architectures, input representations, speech models, training procedures, and testing procedures.

Unless otherwise noted, all experiments in this section were performed with the Resource Management database under the following conditions (see Appendix A for more details):

- Network architecture:
  - 16 LDA (or 26 PLP) input coefficients per frame; 9 frame input window.
  - 100 hidden units.
  - 61 context-independent TIMIT phoneme outputs (1 state per phoneme).
  - all activations = [-1..1], except softmax [0..1] for phoneme layer outputs.
- Training:
  - Training set = 2590 sentences (male), or 3600 sentences (mixed gender).
  - Frames presented in random order; weights updated after each frame.
  - Learning rate schedule = optimized via search (see Section 7.3.4.1).
  - No momentum, no derivative offset.
  - Error criterion = Cross Entropy.
- Testing:
  - Cross validation set = 240 sentences (male), or 390 sentences (mixed).
  - Grammar = word pairs $\Rightarrow$ perplexity 60.
  - One pronunciation per word in the dictionary.
  - Minimum duration constraints for phonemes, via state duplication.
  - Viterbi search, using $\log(Y_i/P_i)$, where P_i = prior of phoneme i.

7.3.1. Network Architectures

The following series of experiments attempt to answer the question: "What is the optimal neural network architecture for frame level training of a speech recognizer?"

7.3.1.1. Benefit of a Hidden Layer

In optimizing the design of a neural network, the first question to consider is whether the network should have a hidden layer, or not. Theoretically, a network with no hidden layers (a *single layer perceptron,* or SLP) can form only linear decision regions, but it is guaranteed to attain 100% classification accuracy if its training set is linearly separable. By contrast, a network with one or more hidden layers (a *multilayer perceptron*, or MLP) can form nonlinear decision regions, but it is liable to get stuck in a local minimum which may be inferior to the global minimum.

It is commonly assumed that an MLP is better than an SLP for speech recognition, because speech is known to be a highly nonlinear domain, and experience has shown that the problem of local minima is insignificant except in artificial tasks. We tested this assumption with a simple experiment, directly comparing an SLP against an MLP containing one hidden layer with 100 hidden units; both networks were trained on 500 training sentences. The MLP achieved 81% word accuracy, while the SLP obtained only 58% accuracy. Thus, a hidden layer is clearly useful for speech recognition.

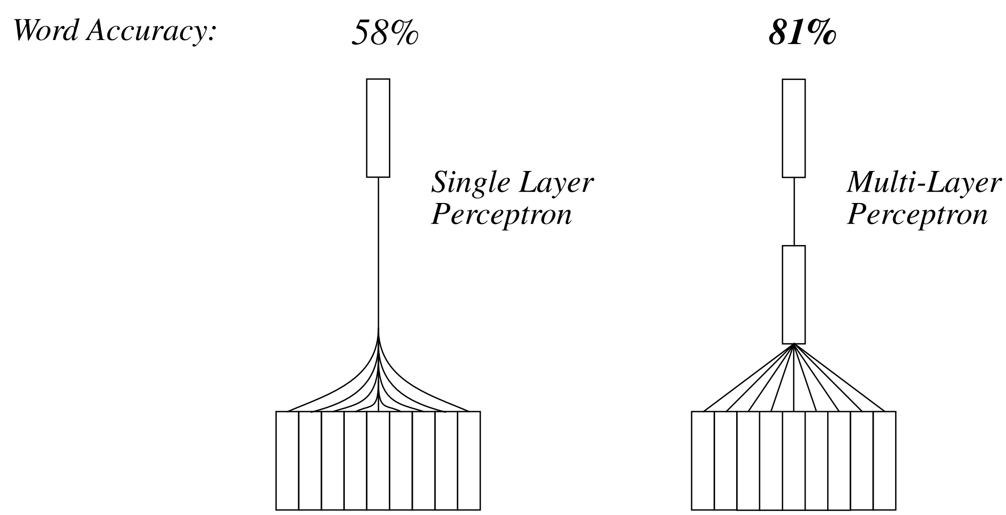


Figure 7.4: A hidden layer is necessary for good word accuracy.

We did not evaluate architectures with more than one hidden layer, because:

1. It has been shown (Cybenko 1989) that any function that can be computed by an MLP with multiple hidden layers can be computed by an MLP with just a single hidden layer, if it has enough hidden units; and
2. Experience has shown that training time increases substantially for networks with multiple hidden layers.

However, it is worth noting that our later experiments with Word Level Training (see Section 7.4) effectively added extra layers to the network.

7.3.1.2. Number of Hidden Units

The number of hidden units has a strong impact on the performance of an MLP. The more hidden units a network has, the more complex decision surfaces it can form, and hence the better classification accuracy it can attain. Beyond a certain number of hidden units, however, the network may possess so much modeling power that it can model the idiosyncrasies of the training data if it's trained too long, undermining its performance on testing data. Common wisdom holds that the optimal number of hidden units should be determined by optimizing performance on a cross validation set.

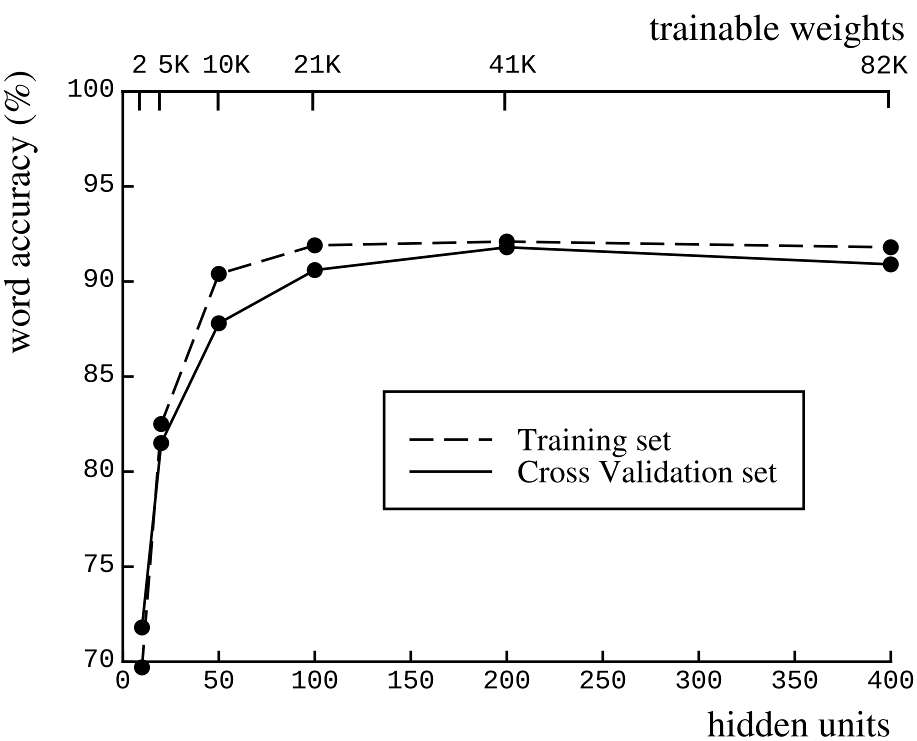


Figure 7.5: Performance improves with the number of hidden units.

Figure 7.5 shows word recognition accuracy as a function of the number of hidden units, for both the training set and the cross validation set. (Actually, performance on the training set was measured on only the first 250 out of the 2590 training sentences, for efficiency.) It can be seen that word accuracy continues to improve on both the training set and the cross validation set as more hidden units are added — at least up to 400 hidden units. This indicates that there is so much variability in speech that it is virtually impossible for a neural network to memorize the training set. We expect that performance would continue to improve beyond 400 hidden units, at a very gradual rate. (Indeed, with the aid of a powerful parallel supercomputer, researchers at ICSI have found that word accuracy continues to improve with as many as 2000 hidden units, using a network architecture similar to ours.) However, because each doubling of the hidden layer doubles the computation time, in the remainder of our experiments we usually settled on 100 hidden units as a good compromise between word accuracy and computational requirements.

7.3.1.3. Size of Input Window

The word accuracy of a system improves with the context sensitivity of its acoustic models. One obvious way to enhance context sensitivity is to show the acoustic model not just one speech frame, but a whole window of speech frames, i.e., the current frame plus the surrounding context. This option is not normally available to an HMM, however, because an HMM assumes that speech frames are mutually independent, so that the only frame that has any relevance is the current frame[1]; an HMM must rely on a large number of context-dependent models instead (such as triphone models), which are trained on single frames from corresponding contexts. By contrast, a neural network can easily look at any number of input frames, so that even context-independent phoneme models can become arbitrarily context sensitive. This means that it should be trivial to increase a network's word accuracy by simply increasing its input window size.

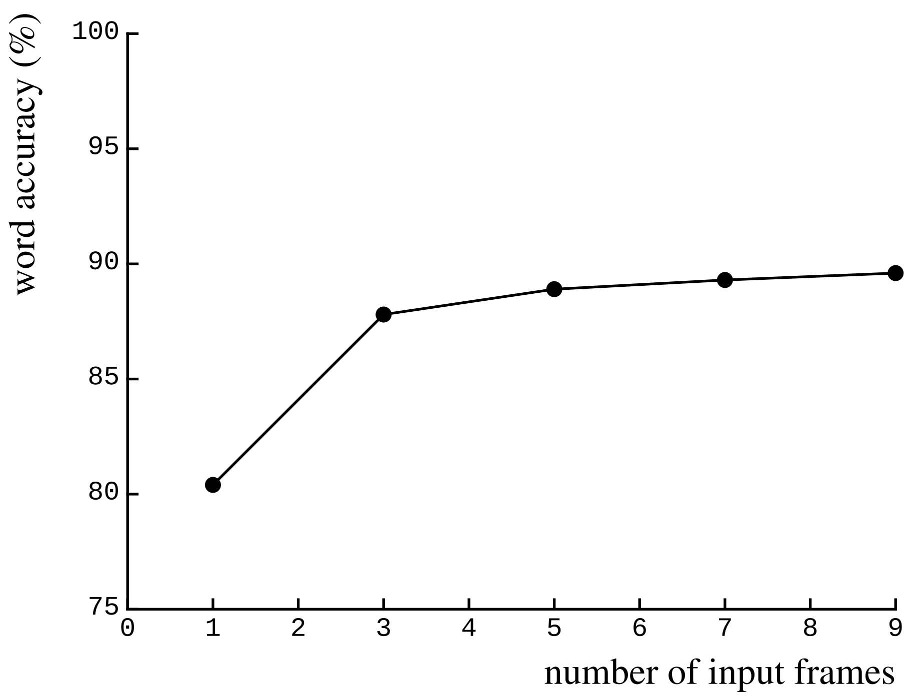


Figure 7.6: Enlarging the input window enhances context sensitivity, and so improves word accuracy.

We tried varying the input window size from 1 to 9 frames of speech, using our MLP which modeled 61 context-independent phonemes. Figure 7.6 confirms that the resulting word accuracy increases steadily with the size of the input window. We expect that the context sensitivity and word accuracy of our networks would continue to increase with more input frames, until the marginal context becomes irrelevant to the central frame being classified.

1. It is possible to get around this limitation, for example by introducing multiple streams of data in which each stream corresponds to another neighboring frame, but such solutions are unnatural and rarely used.

In all of our subsequent experiments, we limited our networks to 9 input frames, in order to balance diminishing marginal returns against increasing computational requirements.

Of course, neural networks can be made not only context-sensitive, but also context-dependent like HMMs, by using any of the techniques described in Sec. 4.3.6. However, we did not pursue those techniques in our research into classification networks, due to a lack of time.

7.3.1.4. Hierarchy of Time Delays

In the experiments described so far, all of the time delays were located between the input window and the hidden layer. However, this is not the only possible configuration of time delays in an MLP. Time delays can also be distributed hierarchically, as in a Time Delay Neural Network. A hierarchical arrangement of time delays allows the network to form a corresponding hierarchy of feature detectors, with more abstract feature detectors at higher layers (Waibel et al, 1989); this allows the network to develop a more compact representation of speech (Lang 1989). The TDNN has achieved such renowned success at phoneme recognition that it is now often assumed that hierarchical delays are necessary for optimal performance. We performed an experiment to test whether this assumption is valid for continuous speech recognition.

We compared three networks, as shown in Figure 7.7:

(a) A simple MLP with 9 frames in the input window, 16 input coefficients per frame, 100 hidden units, and 61 phoneme outputs (20,661 weights total);

(b) An MLP with the same number of input, hidden, and output units as (a), but whose time delays are hierarchically distributed between the two layers (38661 weights);

(c) An MLP like (b), but with only 53 hidden units, so that the number of weights is approximately the same as in (a) (20519 weights).

All three networks were trained on 500 sentences and tested on 60 cross validation sentences. Surprisingly, the best results were achieved by the network without hierarchical delays (although its advantage was not statistically significant). We note that Hild (1994, personal correspondence) performed a similar comparison on a large database of spelled letters, and likewise found that a simple MLP performed at least as well as a network with hierarchical delays.

Our findings seemed to contradict the conventional wisdom that the hierarchical delays in a TDNN contribute to optimal performance. This apparent contradiction is resolved by noting that the TDNN's hierarchical design was initially motivated by a poverty of training data (Lang 1989); it was argued that the hierarchical structure of a TDNN leads to replication of weights in the hidden layer, and these replicated weights are then trained on shifted subsets of the input speech window, effectively increasing the amount of training data per weight, and improving generalization to the testing set. Lang found hierarchical delays to be essential for coping with his tiny database of 100 training samples per class ("B, D, E, V"); Waibel et al (1989) also found them to be valuable for a small database of about 200 samples per class (/`b,d,g`/). By contrast, our experiments (and Hild's) used over 2,700 train-

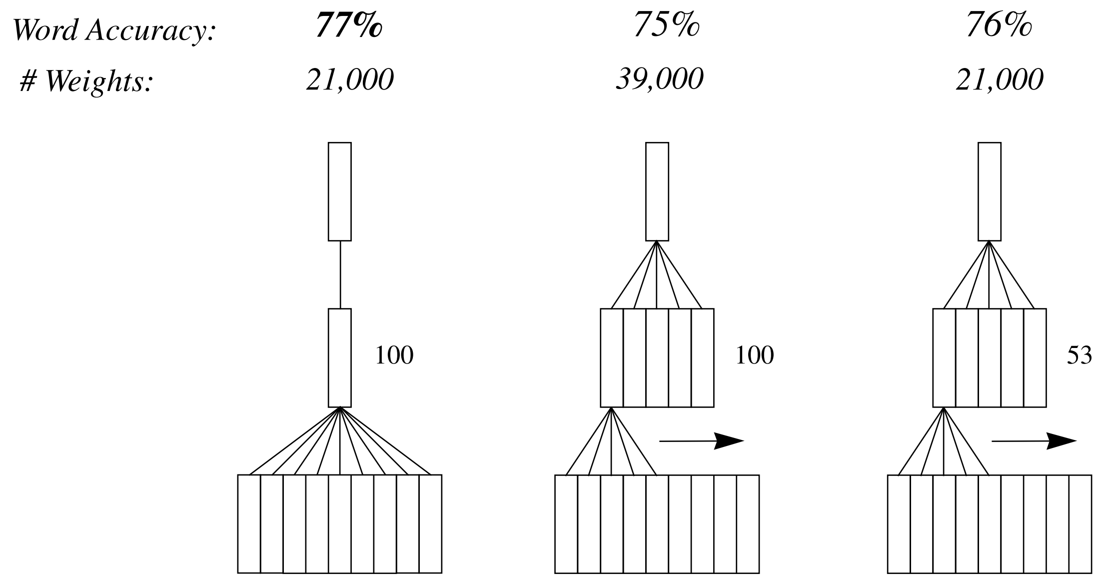


Figure 7.7: Hierarchical time delays do not improve performance when there is abundant training data.

ing samples per class. Apparently, when there is such an abundance of training data, it is no longer necessary to boost the amount of training data per weight via hierarchical delays.

In fact, it can be argued that for a large database, hierarchical delays will theoretically degrade system performance, due to an inherent tradeoff between the degree of hierarchy and the trainability of a network. As time delays are redistributed higher within a network, each hidden unit sees less context, so it becomes a simpler, less potentially powerful pattern recognizer; however, as we have seen, it also receives more training, because it is applied over several adjacent positions, with tied weights, so it learns its simpler patterns more reliably. Consequently, when relatively little training data is available, hierarchical time delays serve to increase the amount of training data per weight and improve the system's accuracy; but when a large amount of training data is available, a TDNN's hierarchical time delays make the hidden units unnecessarily coarse and hence degrade the system's accuracy, so a simple MLP becomes theoretically preferable. This seems to be what we observed in our experiment with a large database.

7.3.1.5. Temporal Integration of Output Activations

A TDNN is distinguished from a simple MLP not only by its hierarchical time delays, but also by the temporal integration of phoneme activations over several time delays. Lang (1989) and Waibel et al (1989) argued that temporal integration makes the TDNN time-shift invariant, i.e., the TDNN is able to classify phonemes correctly even if they are poorly segmented, because the TDNN's feature detectors are finely tuned for shorter segments, and will contribute to the overall score no matter where they occur within a phonemic segment.

Although temporal integration was clearly useful for phoneme classification, we wondered whether it was still useful for continuous speech recognition, given that temporal inte-

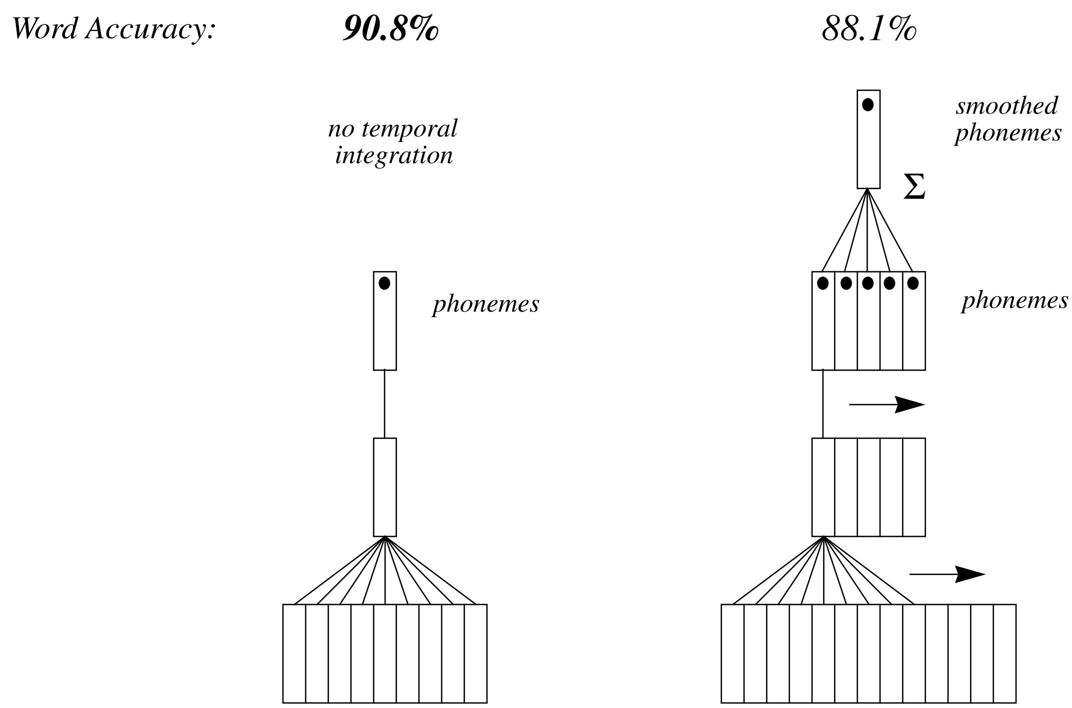


Figure 7.8: Temporal integration of phoneme outputs is redundant and not helpful.

gration is now performed by DTW over the whole utterance. We did an experiment to compare the word accuracy resulting from the two architectures shown in Figure 7.8. The first network is a standard MLP; the second network is an MLP whose phoneme level activations are summed over 5 frames and then normalized to yield smoothed phoneme activations. In each case, we trained the network on data centered on each frame within the whole database, so there was no difference in the prior probabilities. Each network used softmax activations in its final layer, and tanh activations in all preceding layers. We emphasize that temporal integration was performed twice in the second system — once by the network itself, in order to smooth the phoneme activations, and later by DTW in order to determine a score for the whole utterance. We found that the simple MLP achieved 90.8% word accuracy, while the network with temporal integration obtained only 88.1% word accuracy. We conclude that TDNN-style temporal integration of phoneme activations is counterproductive for continuous speech recognition, because it is redundant with DTW, and also because such temporally smoothed phoneme activations are blurrier and thus less useful for DTW.

7.3.1.6. Shortcut Connections

It is sometimes argued that direct connections from the input layer to the output layer, bypassing the hidden layer, can simplify the decision surfaces found by a network, and thus improve its performance. Such *shortcut connections* would appear to be more promising for predictive networks than for classification networks, since there is a more direct relationship between inputs and outputs in a predictive network. Nevertheless, we performed a simple

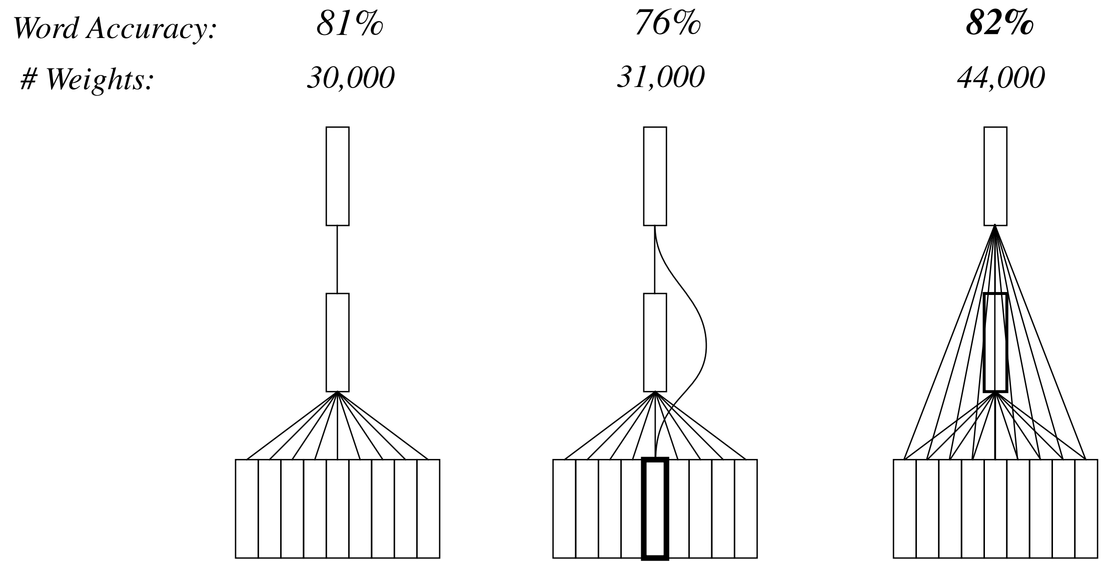


Figure 7.9: Shortcut connections have an insignificant advantage, at best.

experiment to test this idea for our classification network. We compared three networks, as shown in Figure 7.9:

(a) a standard MLP with 9 input frames;

(b) an MLP augmented by a direct connection from the central input frame to the current output frame;

(c) an MLP augmented by direct connections from all 9 input frames to the current output frame.

All three networks were trained on 500 sentences and tested on 60 cross validation sentences. Network (c) achieved the best results, by an insignificantly small margin. It was not surprising that this network achieved slightly better performance than the other two networks, since it had 50% more weights as a result of all of its shortcut connections. We conclude that the intrinsic advantage of shortcut connections is negligible, and may be attributed merely to the addition of more parameters, which can be achieved just as easily by adding more hidden units.

7.3.1.7. Transfer Functions

The choice of transfer functions (which convert the net input of each unit to an activation value) can make a significant difference in the performance of a network. Linear transfer functions are not very useful since multiple layers of linear functions can be collapsed into a single linear function; hence they are rarely used, especially below the output layer. By contrast, nonlinear transfer functions, which squash any input into a fixed range, are much more powerful, so they are used almost exclusively. Several popular nonlinear transfer functions are shown in Figure 7.10.

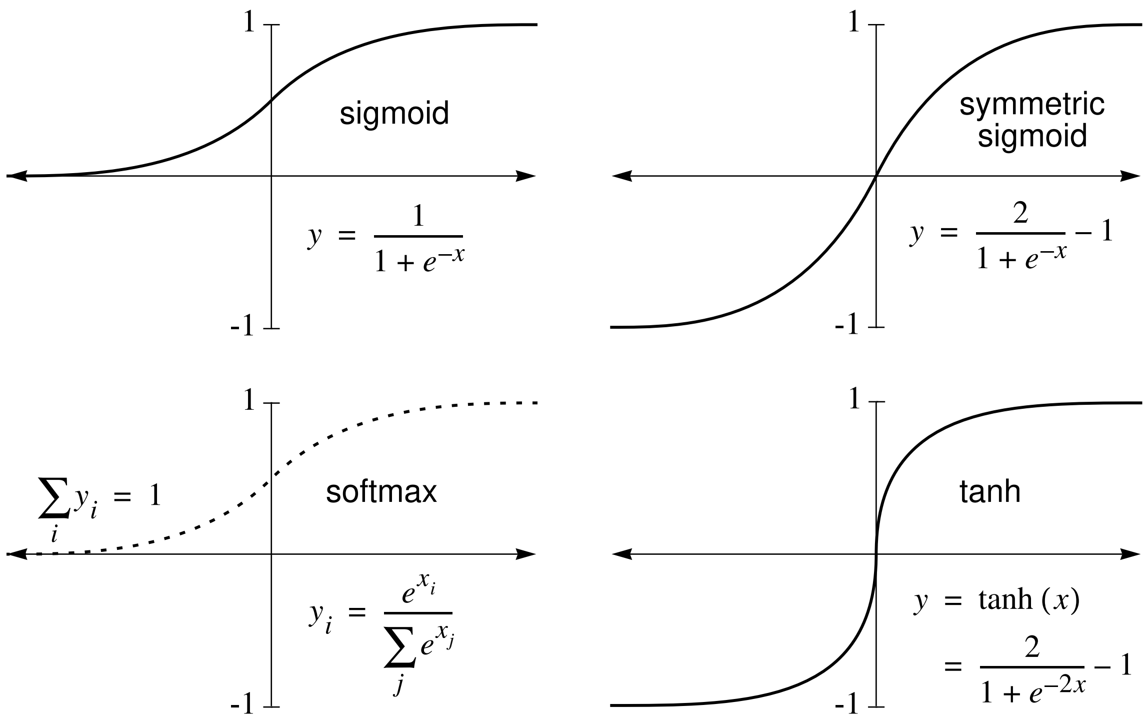


Figure 7.10: Four popular transfer functions, for converting a unit's net input *x* to an activation *y*.

The sigmoid function, which has an output range [0,1], has traditionally served as the "default" transfer function in neural networks. However, the sigmoid has the disadvantage that it gives a nonzero mean activation, so that the network must waste some time during early training just pushing its biases into a useful range. It is now widely recognized that networks learn most efficiently when they use *symmetric* activations (i.e., in the range [-1,1]) in all non-output units (including the input units), hence the symmetric sigmoid or tanh functions are often preferred over the sigmoid function. Meanwhile, the *softmax* function has the special property that it constrains all the activations to sum to 1 in any layer where it is applied; this is useful in the output layer of a classification network, because the output activations are known to be estimate of the posterior probabilities $P(class|input)$, which should add up to 1. (We note, however, that even without this constraint, our networks' outputs typically add up to something in the range of 0.95 to 1.05, if each output activation is in the range [0,1].)

Based on these considerations, we chose to give each network layer its own transfer function, so that we could use the softmax function in the output layer, and a symmetric or tanh function in the hidden layer (we also normalized our input values to lie within the range [-1,1]). Figure 7.11 shows the learning curve of this "standard" set of transfer functions (solid line), compared against that of two other configurations. (In these experiments, performed at an early date, we trained on frames in sequential order within each of 3600 training sentences, updating the weights after each sentence; and we used a fixed, geometrically decreasing learning rate schedule.) These curves confirm that performance is much better when the hidden layer uses a symmetric function (tanh) rather than the sigmoid function.

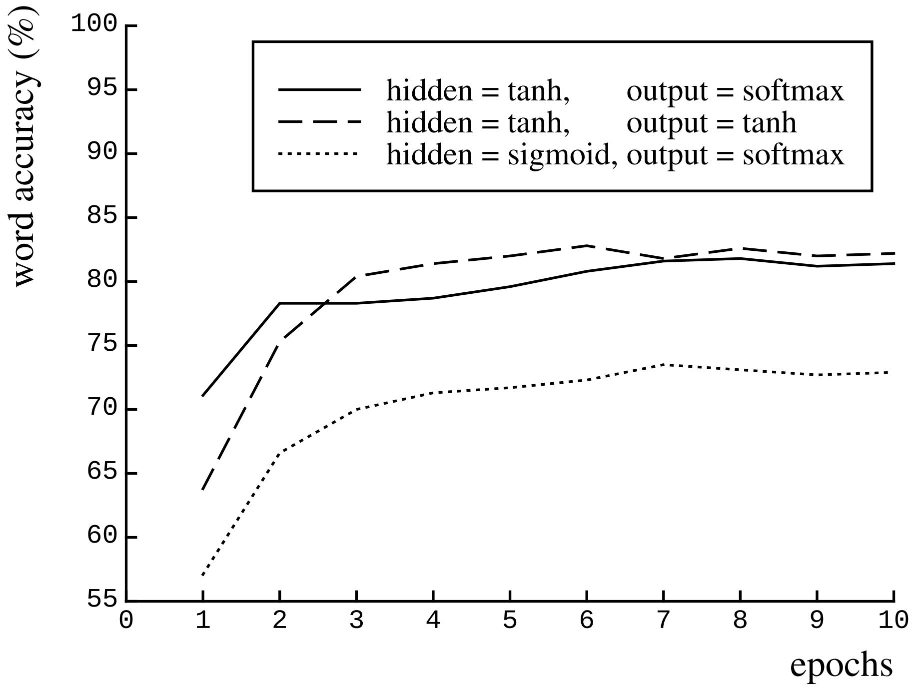


Figure 7.11: Results of training with different transfer functions in the hidden and output layers.

Also, we see that learning is accelerated when the output layer uses the softmax function rather than an unconstrained function (tanh), although there is no statistically significant difference in their performance in the long run.

7.3.2. Input Representations

It is universally agreed that speech should be represented as a sequence of frames, resulting from some type of signal analysis applied to the raw waveform. However, there is no universal agreement as to which type of signal processing ultimately gives the best performance; the optimal representation seems to vary from system to system. Among the most popular representations, produced by various forms of signal analysis, are spectral (FFT) coefficients, cepstral (CEP) coefficients, linear predictive coding (LPC) coefficients, and perceptual linear prediction (PLP) coefficients. Since every representation has its own champions, we did not expect to find much difference between the representations; nevertheless, we felt obliged to compare some of these representations in the environment of our NN-HMM hybrid system.

We studied the following representations (with a 10 msec frame rate in each case):

- **FFT-16:** 16 melscale spectral coefficients per frame. These coefficients, produced by the Fast Fourier Transform, represent discrete frequencies, distributed linearly in the low range but logarithmically in the high range, roughly corresponding to the ranges of sensitivity in the human ear. Adjacent spectral coefficients are mutually correlated; we imagined that this might simplify the pattern recognition task for a neural network. Viewed over time, spectral coefficients form a spectrogram (as in Figure 6.5), which can be interpreted visually.

- **FFT-32:** 16 melscale spectral coefficients augmented by their first order differences (between *t*-2 and *t*+2). The addition of delta information makes explicit what is already implicit in a window of FFT-16 frames. We wanted to see whether this redundancy is useful for a neural network, or not.
- **LDA-16:** Compression of FFT-32 into its 16 most significant dimensions, by means of linear discriminant analysis. The resulting coefficients are uncorrelated and visually uninterpretable, but they are dense in information content. We wanted to see whether our neural networks would benefit from such compressed inputs.
- **PLP-26:** 12 perceptual linear prediction coefficients augmented by the frame's power, and the first order differences of these 13 values. PLP coefficients are the cepstral coefficients of an autoregressive all-pole model of a spectrum that has been specially enhanced to emphasize perceptual features (Hermansky 1990). These coefficients are uncorrelated, so they cannot be interpreted visually.

All of these coefficients lie in the range [0,1], except for the PLP-26 coefficients, which had irregular ranges varying from [-.5,.5] to [-44,44] because of the way they were normalized in the package that we used.

7.3.2.1. Normalization of Inputs

Theoretically, the range of the input values should not affect the asymptotic performance of a network, since the network can learn to compensate for scaled inputs with inversely scaled weights, and it can learn to compensate for a shifted mean by adjusting the bias of the hidden units. However, it is well known that networks learn more efficiently if their inputs are all normalized in the same way, because this helps the network to pay equal attention to every input. Moreover, the network also learns more efficiently if the inputs are normalized to be symmetrical around 0, as explained in Section 7.3.1.7. (In an early experiment, symmetrical [-1..1] inputs achieved 75% word accuracy, while asymmetrical [0..1] inputs obtained only 42% accuracy.)

We studied the effects of normalizing the PLP coefficients to a mean of 0 and standard deviation of σ for different values of σ, comparing these representations against PLP inputs without normalization. In each case, the weights were randomly initialized to the same range, $\pm 1 / \sqrt{\text{fanin}}$. For each input representation, we trained on 500 sentences and tested on 60 cross validation sentences, using a learning rate schedule that was separately optimized for each case. Figure 7.12 shows that the learning curves are strongly affected by the standard deviation. On the one hand, when $\sigma \geq 1$, learning is erratic and performance remains poor for many iterations. This apparently occurs because large inputs lead to large net inputs into the hidden layer, causing activations to saturate, so that their derivatives remain small and learning takes place very slowly. On the other hand, when $\sigma \leq 0.5$, we see that normalization is extremely valuable. $\sigma = 0.5$ gave slightly better asymptotic

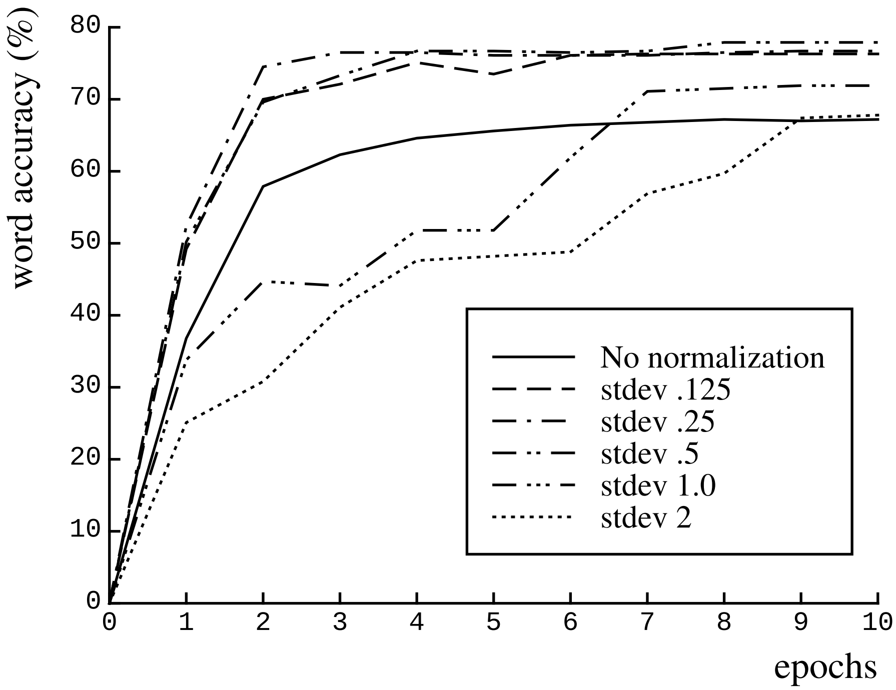


Figure 7.12: Normalization of PLP inputs is very helpful.

results than $\sigma < 0.5$, so we used $\sigma = 0.5$ for subsequent experiments. Of course, this optimal value of σ would be twice as large if the initial weights were twice as small, or if the sigmoidal transfer functions used in the hidden layer (tanh) were only half as steep.

We note that $\sigma = 0.5$ implies that 95% of the inputs lie in the range [-1,1]. We found that saturating the normalized inputs at [-1,1] did not degrade performance, suggesting that such extreme values are semantically equivalent to ceilinged values. We also found that quantizing the input values to 8 bits of precision did not degrade performance. Thus, we were able to conserve disk space by encoding each floating point input coefficient (in the range [-1,1]) as a single byte in the range [0..255], with no loss of performance.

Normalization may be based on statistics that are either *static* (collected from the entire training set, and kept constant during testing), or *dynamic* (collected from individual sentences during both training and testing). We compared these two methods, and found that it makes no significant difference which is used, as long as it is used consistently. Performance erodes only if these methods are used inconsistently during training and testing. For example, in an experiment where training used static normalization, word accuracy was 90% if testing also used static normalization, but only 84% if testing used dynamic normalization. Because static and dynamic normalization gave equivalent results when used consistently, we conclude that dynamic normalization is preferable only if there is any possibility that the training and testing utterances were recorded under different conditions (such that static statistics do not apply to both).

7.3.2.2. Comparison of Input Representations

In order to make a fair comparison between our four input representations, we first normalized all of them to the same symmetric range, [-1,1]. Then we evaluated a network on each representation, using an input window of 9 frames in each case; these networks were trained on 3600 sentences and tested on 390 sentences. The resulting learning curves are shown in Figure 7.13.

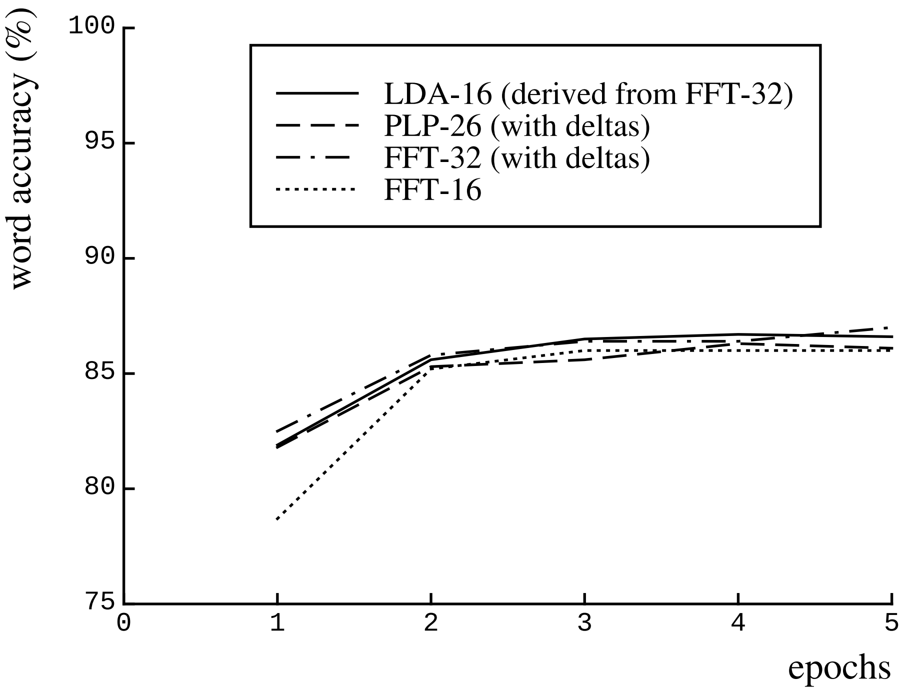


Figure 7.13: Input representations, all normalized to [-1..1]: Deltas and LDA are moderately useful.

The most striking observation is that FFT-16 gets off to a relatively slow start, because given this representation the network must automatically discover the temporal dynamics implicit in its input window, whereas the temporal dynamics are explicitly provided in the other representations (as delta coefficients). Although this performance gap shrinks over time, we conclude that delta coefficients are nevertheless moderately useful for neural networks.

There seems to be very little difference between the other representations, although PLP-26 coefficients may be slightly inferior. We note that there was no loss in performance from compressing FFT-32 coefficients into LDA-16 coefficients, so that LDA-16 was always better than FFT-16, confirming that it is not the number of coefficients that matters, but their information content. We conclude that LDA is a marginally useful technique because it orthogonalizes and reduces the dimensionality of the input space, making the computations of the neural network more efficient.

7.3.3. Speech Models

Given enough training data, the performance of a system can be improved by increasing the specificity of its speech models. There are many ways to increase the specificity of speech models, including:

- augmenting the number of phones (e.g., by splitting the phoneme /b/ into /b:closure/ and /b:burst/, and treating these independently in the dictionary of word pronunciations);
- increasing the number of states per phone (e.g., from 1 state to 3 states for every phone);
- making the phones context-dependent (e.g., using diphone or triphone models);
- modeling variations in the pronunciations of words (e.g., by including multiple pronunciations in the dictionary).

Optimizing the degree of specificity of the speech models for a given database is a time-consuming process, and it is not specifically related to neural networks. Therefore we did not make a great effort to optimize our speech models. Most of our experiments were performed using 61 context-independent TIMIT phoneme models, with a single state per phoneme, and only a single pronunciation per word. We believe that context-dependent phone models would significantly improve our results, as they do for HMMs; but we did not have time to explore them. We did study a few other variations on our speech models, however, as described in the following sections.

7.3.3.1. Phoneme Topology

Most of our experiments used a single state per phoneme, but at times we used up to 3 states per phoneme, with simple left-to-right transitions. In one experiment, using 3600 training sentences and 390 cross validation sentences, we compared three topologies:

- 1 state per phoneme;
- 3 states per phoneme;
- between 1 and 3 states per phoneme, according to the minimum encountered duration of that phoneme in the training set.

Figure 7.14 shows that best results were obtained with 3 states per phoneme, and results deteriorated with fewer states per phoneme. Each of these experiments used the same minimum phoneme duration constraints (the duration of each phoneme was constrained, by means of state duplication, to be at least 1/2 the average duration of that phoneme as measured in the training set); therefore the fact that the 1...3 state model outperformed the 1 state model was not simply due to better duration modeling, but due to the fact that the additional states per phoneme were genuinely useful, and that they received adequate training.

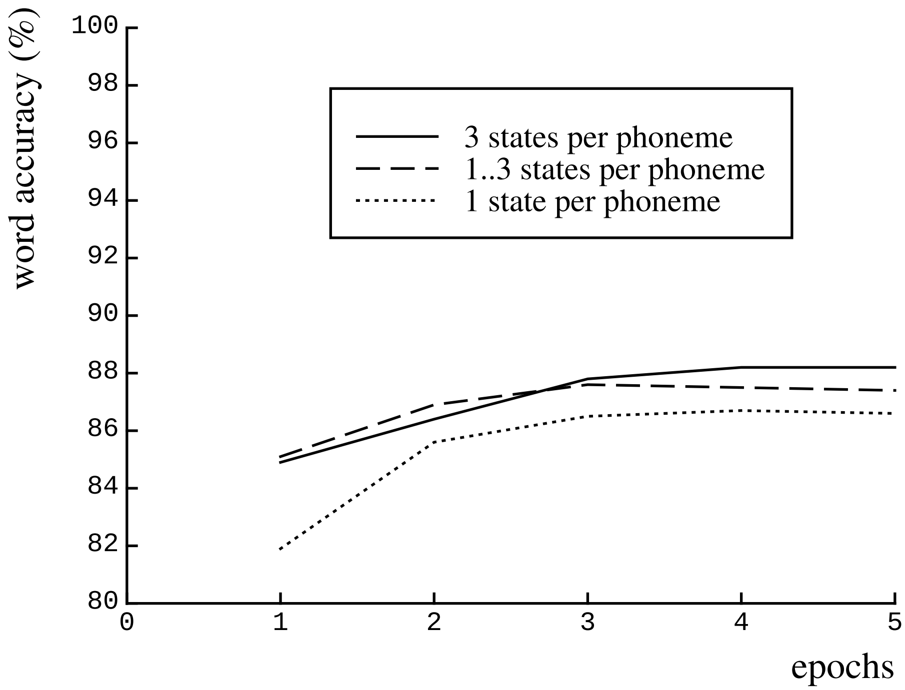


Figure 7.14: A 3-state phoneme model outperforms a 1-state phoneme model.

7.3.3.2. Multiple Pronunciations per Word

It is also possible to improve system performance by making the dictionary more flexible, e.g., by allowing multiple pronunciations per word. We tried this technique on a small scale. Examining the results of a typical experiment, we found that the words "a" and "the" caused more errors than any other words. This was not surprising, because these words are ubiquitous and they each have at least two common pronunciations (with short or long vowels), whereas the dictionary listed only one pronunciation per word. Thus, for example, the word "the" was often misrecognized as "me", because the dictionary only provided "the" with a short vowel (/DX AX/).

We augmented our dictionary to include both the long and short pronunciations for the words "a" and "the", and retested the system. We found that this improved the word accuracy of the system from 90.7% to 90.9%, by fixing 11 errors while introducing 3 new errors that resulted from confusions related to the new pronunciations. While it may be possible to significantly enhance a system's performance by a systematic optimization of the dictionary, we did not pursue this issue any further, considering it outside the scope of this thesis.

7.3.4. Training Procedures

We used backpropagation to train all of our networks, but within that framework we explored many variations on the training procedure. In this section we present our research on training procedures, including learning rate schedules, momentum, data presentation and update schedules, gender dependent training, and recursive labeling.

7.3.4.1. Learning Rate Schedules

The learning rate schedule is of critical importance when training a neural network. If the learning rate is too small, the network will converge very slowly; but if the learning rate is too high, the gradient descent procedure will overshoot the downward slope and enter an upward slope instead, so the network will oscillate. Many factors can affect the optimal learning rate schedule of a given network; unfortunately there is no good understanding of what those factors are. If two dissimilar networks are trained with the same learning rate schedule, it will be unfair to compare their results after a fixed number of iterations, because the learning rate schedule may have been optimal for one of the networks but suboptimal for the other. We eventually realized that many of the conclusions drawn from our early experiments were invalid for this reason.

Because of this, we finally decided to make a systematic study of the effect of learning rate schedules on network performance. In most of these experiments we used our standard network configuration, training on 3600 sentences and cross validating on 60 sentences. We began by studying constant learning rates. Figure 7.15 shows the learning curves (in terms of both frame accuracy and word accuracy) that resulted from constant learning rates in the range .0003 to .01. We see that a learning rate of .0003 is too small (word accuracy is still just 10% after the first iteration of training), while .01 is too large (both frame and word accuracy remain suboptimal because the network is oscillating). Meanwhile, a learning rate of .003 gave best results at the beginning, but .001 proved better later on. From this we conclude that the learning rate should decrease over time, in order to avoid disturbing the network too much as it approaches the optimal solution.

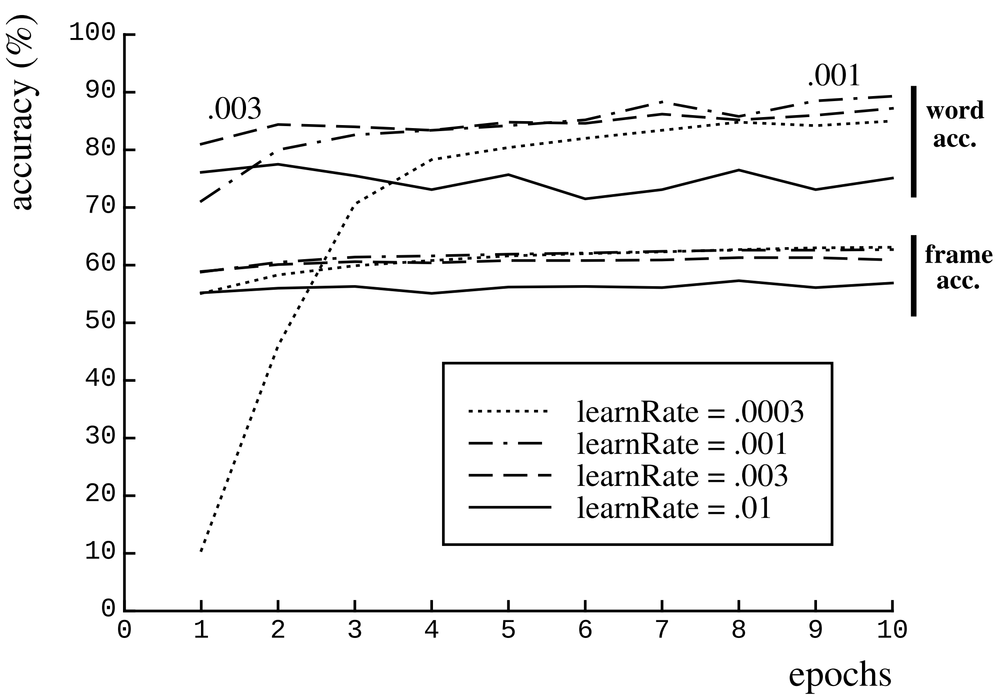


Figure 7.15: Constant learning rates are unsatisfactory; the learning rate should decrease over time.

The next question is, exactly how should the learning rate shrink over time? We studied schedules where the learning rate starts at .003 (the optimal value) and then shrinks geometrically, by multiplying it by some constant factor less than 1 after each iteration of training. Figure 7.16 shows the learning rates that resulted from geometric factors ranging from 0.5 to 1.0. We see that a factor of 0.5 (i.e., halving the learning rate after each iteration) initially gives the best frame and word accuracy, but this advantage is soon lost, because the learning rate shrinks so quickly that the network cannot escape from local minima that it wanders into. Meanwhile, as we have already seen, a factor of 1.0 (a constant learning rate) causes the learning rate to remain too large, so learning is unstable. The best geometric factor seems to be an intermediate value of 0.7 or 0.8, which gives the network time to escape from local minima before the learning rate effectively shrinks to zero.v

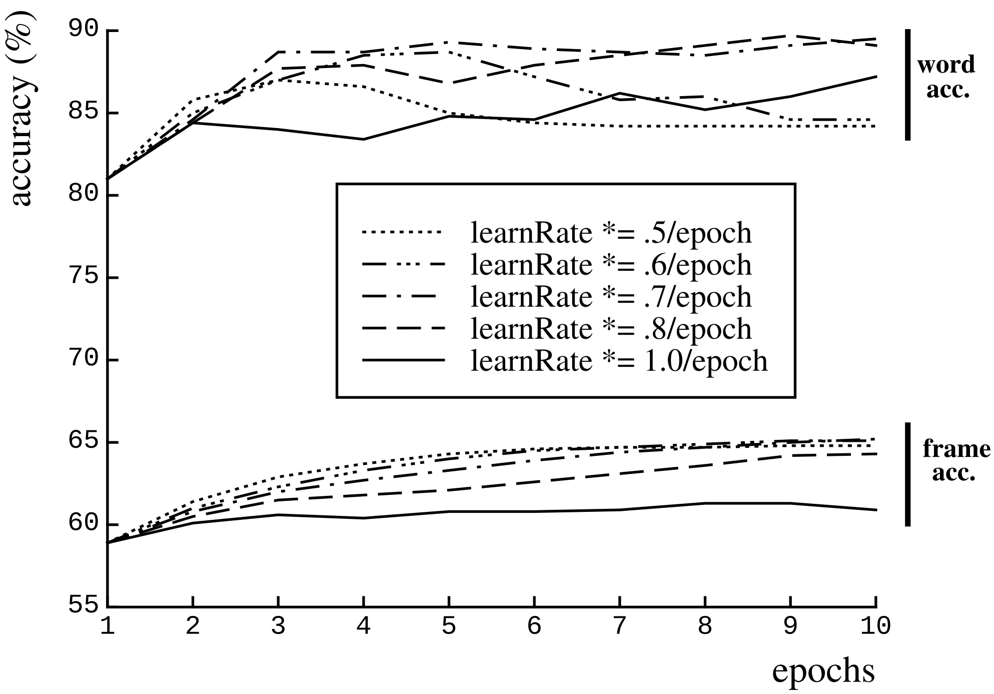


Figure 7.16: Geometric learning rates (all starting at LR = .003) are better, but still may be suboptimal.

Although a geometric learning rate schedule is clearly useful, it may still be suboptimal. How do we know that a network really learned as much as it could before the learning rate vanished? And isn't it possible that the learning rate should shrink nongeometrically, for example, shrinking by 60% at first, and later only by 10%? And most importantly, what guarantee is there that a fixed learning rate schedule that has been optimized for one set of conditions will still be optimal for another set of conditions? Unfortunately, there is no such guarantee.

Therefore, we began studying learning rate schedules that are based on dynamic search. We developed a procedure that repeatedly searches for the optimal learning rate during each iteration; the algorithm is as follows. Beginning with an initial learning rate in iteration #1, we train for one iteration and measure the cross validation results. Then we start over and train for one iteration again, this time using half the learning rate, and again measure the cross validation results. Comparing these two results, we can infer whether the optimal learning rate for iteration #1 is larger or smaller than these values, and accordingly we either double or halve the nearest learning rate, and try again. We continue doubling or halving the learning rate in this way until the accuracy finally gets worse for some learning rate. Next we begin interpolating between known points (x = learning rate, y = accuracy), using a quadratic interpolation on the best data point and its left and right neighbor, to find successive learning rates to try. That is, if the three best points are (x_1, y_1), (x_2, y_2), and (x_3, y_3), such that the learning rate x_2 gave the best result y_2, then we first solve for the parabola $y = ax^2 + bx + c$ that goes through these three points using Kramer's Rule:

$$a = \frac{\begin{vmatrix} y_1 & x_1 & 1 \\ y_2 & x_2 & 1 \\ y_3 & x_3 & 1 \end{vmatrix}}{D} \qquad b = \frac{\begin{vmatrix} x_1^2 & y_1 & 1 \\ x_2^2 & y_2 & 1 \\ x_3^2 & y_3 & 1 \end{vmatrix}}{D} \qquad c = \frac{\begin{vmatrix} x_1^2 & x_1 & y_1 \\ x_2^2 & x_2 & y_2 \\ x_3^2 & x_3 & y_3 \end{vmatrix}}{D} \qquad D = \begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix}$$

and then we find the highest point of this parabola,

$$(\hat{x}, \hat{y}) = \left(\frac{-b}{2a}, \frac{4ac - b^2}{4a}\right) \tag{74}$$

so that $\hat{x}$ is the next learning rate to try. The search continues in this way until the expected improvement $(\hat{y} - y_2)$ is less than a given threshold, at which point it becomes a waste of time to continue refining the learning rate for iteration #1. (If two learning rates result in indistinguishable performance, we keep the smaller one, because it is likely to be preferable during the next iteration.) We then move on to iteration #2, setting its initial learning rate set to the optimal learning rate from iteration #1, and we begin a new round of search.

We note in passing that it is very important for the search criterion to be the same as the testing criterion. In an early experiment, we compared the results of two different searches, based on either word accuracy or frame accuracy. The search based on word accuracy yielded 65% word accuracy, but the search based on frame accuracy yielded only 48% word accuracy. This discrepancy arose partly because improvements in frame accuracy were too small to be captured by the 2% threshold, so the learning rate rapidly shrank to zero; but it was also partly due to the fact that the search criterion was inconsistent with and poorly correlated with the testing criterion. All of our remaining experiments were performed using word accuracy as the search criterion.

Because the search procedure tries several different learning rates during each iteration of training, this procedure obviously increases the total amount of computation, by a factor that depends on the arbitrary threshold. We typically set the threshold to a 2% relative margin, such that computation time typically increased by a factor of 3-4.

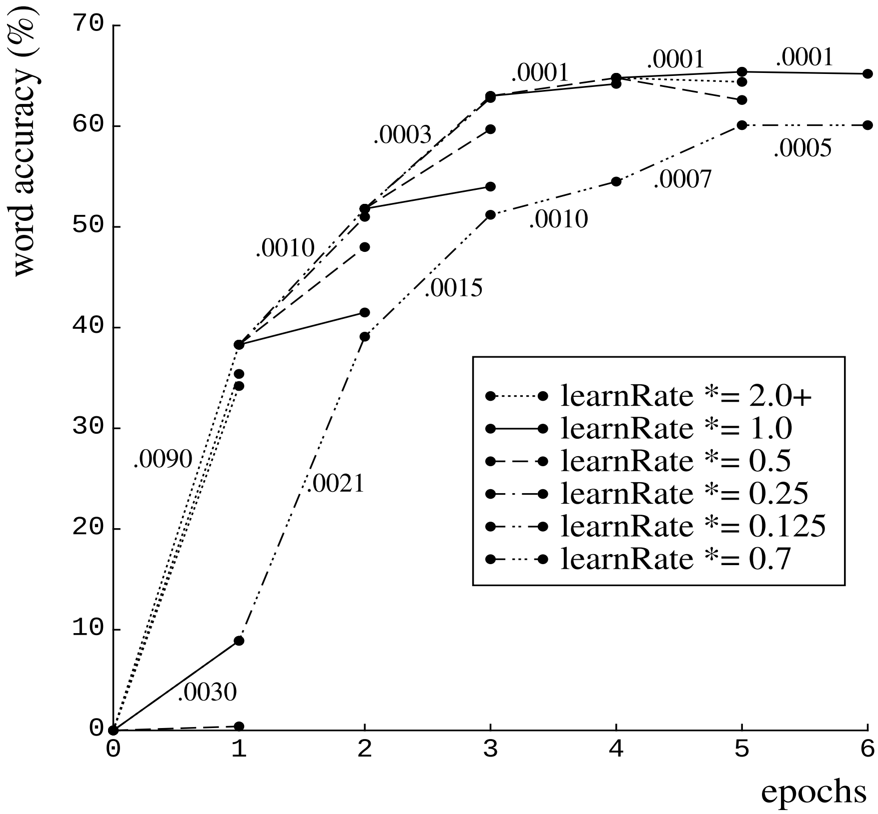


Figure 7.17: Searching for the optimal learning rate schedule.

Figure 7.17 illustrates the search procedure, and its advantage over a geometric schedule. Since the search procedure increases the computation time, we performed this experiment using only 500 training sentences. The lower learning curve in Figure 7.17 corresponds to a fixed geometric schedule with a factor of 0.7 (recall that this factor was optimized on the full training set). The upper learning curves correspond to the search procedure. Different types of lines correspond to different multiplicative factors that were tried during the search procedure; for example, a solid line corresponds to a factor of 1.0 (i.e., same learning rate as in the previous iteration), and a dashed line corresponds to a factor of 0.5 (i.e., half the learning rate as in the previous iteration). The numbers along the upper and lower curves indicate the associated learning rate during each iteration. Several things are apparent from this graph:

- The search procedure gives significantly better results than the geometric schedule. Indeed, the search procedure can be trusted to find a schedule that is nearly optimal in any situation, outperforming virtually any fixed schedule, since it is adaptive.
- The initial learning rate of .003, which was optimal in an earlier experiment, is not optimal anymore, because the experimental conditions have changed (in this case, the number of training sentences has decreased). Because performance is so sensitive to the learning rate schedule, which in turn is so sensitive to experimental conditions, we conclude that it can be very misleading to compare the results of two experiments that were performed under different conditions but which used the

same fixed learning rate schedule. We realized in hindsight that many of our early experiments (not reported in this thesis) were flawed and inconclusive for this reason. This reinforces the value of dynamically searching for the optimal learning rate schedule in every experiment.

- The optimal learning rate schedule starts at .009 and decreases very rapidly at first, but ultimately asymptotes at .0001 as the word accuracy also asymptotes. (Notice how much worse is the accuracy that results from a learning rate multiplied by a constant 1.0 factor [solid lines] or even a 0.5 factor [dashed lines], compared to the optimal factor, during the early iterations.)

The fact that the optimal learning rate schedule decreases asymptotically suggested one more type of fixed learning rate schedule — one that decays asymptotically, as a function of the cross validation performance. We hypothesized a learning rate schedule of the form

$$lr = lr_0 \cdot wordErr^k \tag{75}$$

where lr_0 is the initial learning rate (determined by search), $wordErr$ is the word error rate on the cross validation set (between 0.0 and 1.0), and k is a constant power. Note that this schedule begins with lr_0; it asymptotes whenever the cross validation performance asymptotes; the asymptotic value can be controlled by k; and if $wordErr = 0$, then we also have $lr = 0$. We performed a few experiments with this learning rate schedule (using $k = 5$ to approximate the above optimized schedule); but since this sort of asymptotic schedule appeared less reliable than the geometric schedule, we didn't pursue it very far.

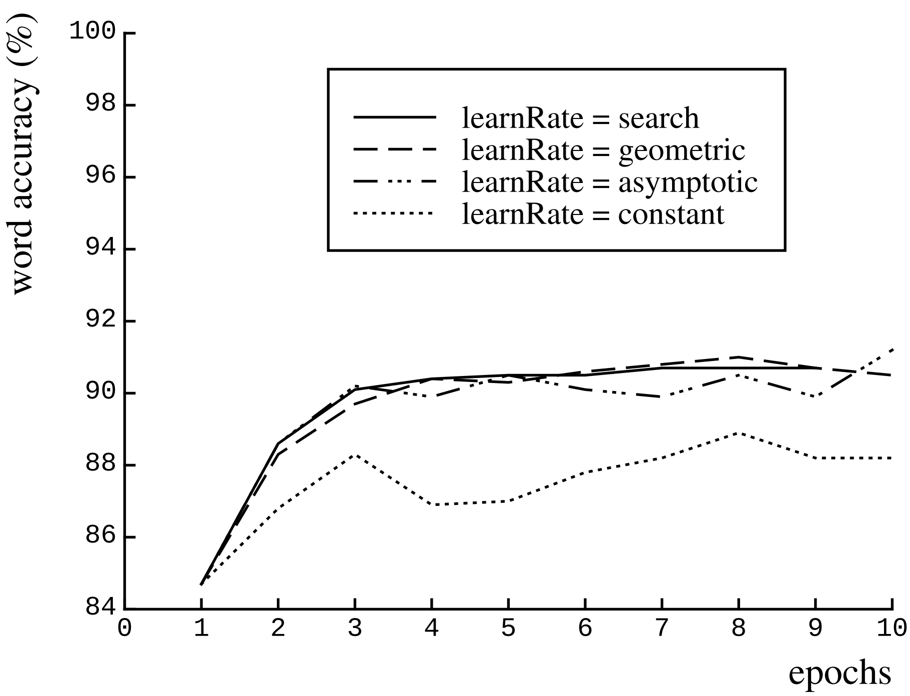


Figure 7.18: Performance of different types of learning rate schedules: Search is reliably optimal.

Figure 7.18 directly compares the performance of each of the above four learning rate schedules — constant, geometric, search, and asymptotic — using a training set of 2590 male sentences, and a cross validation set of 240 male sentences. All four schedules start with the optimal initial learning rate of .01. We see that a constant learning rate schedule (.01) yields the worst performance, because this value remains too high and causes the network to oscillate after the first iteration. The asymptotic schedule begins as optimally as the search schedule, because its learning rate immediately shrinks by more than half; but its later performance is erratic, because its asymptotic learning rate (.00134) is still too high, due to a poor choice of k. The best performance is given by either the search or geometric schedule. Note that the gap between the search and geometric schedules, so wide in the earlier experiment, has now virtually disappeared, because we carefully initialized the geometric schedule with an optimal learning rate this time.

By comparing the various learning rate schedules discovered by our search procedure under different conditions, we have observed that the optimal learning rate schedule is affected by at least the following factors:

- **Number of training sentences.** A larger training set implies smaller learning rates in each iteration, as shown in Figure 7.19. This is primarily because the optimal learning rate curve decays a little after each weight update, and larger training sets travel further down this curve during each iteration, as shown in Figure 7.20. (Interestingly, Figure 7.20 also suggests that learning might be more efficient if we adjusted the learning rate after each sample, rather than after each iteration; but we did not have time to explore this idea.)

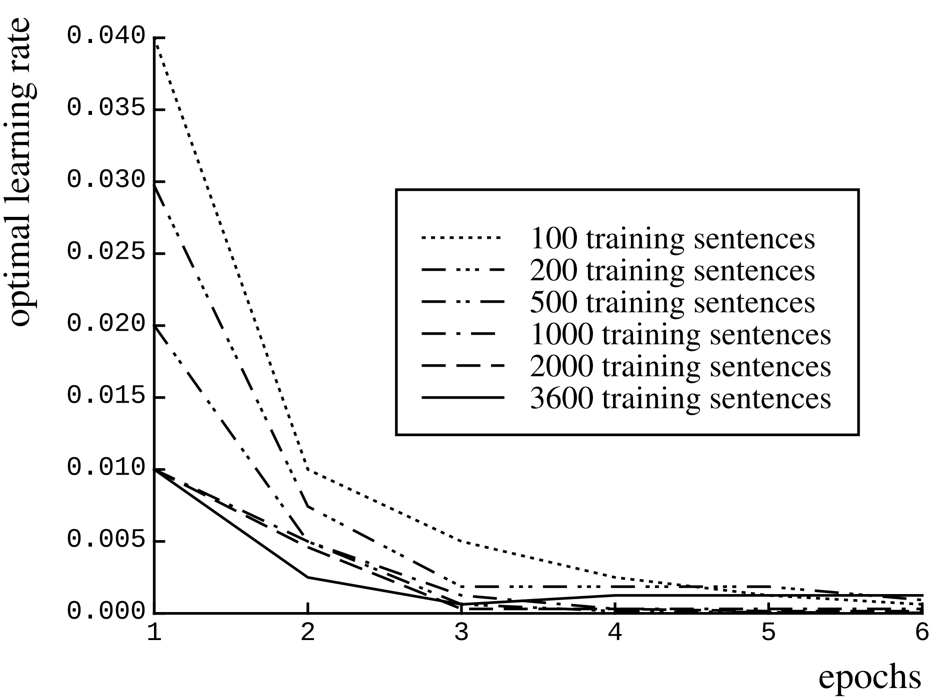


Figure 7.19: Learning rate schedules (as a function of training set size), as optimized by search.

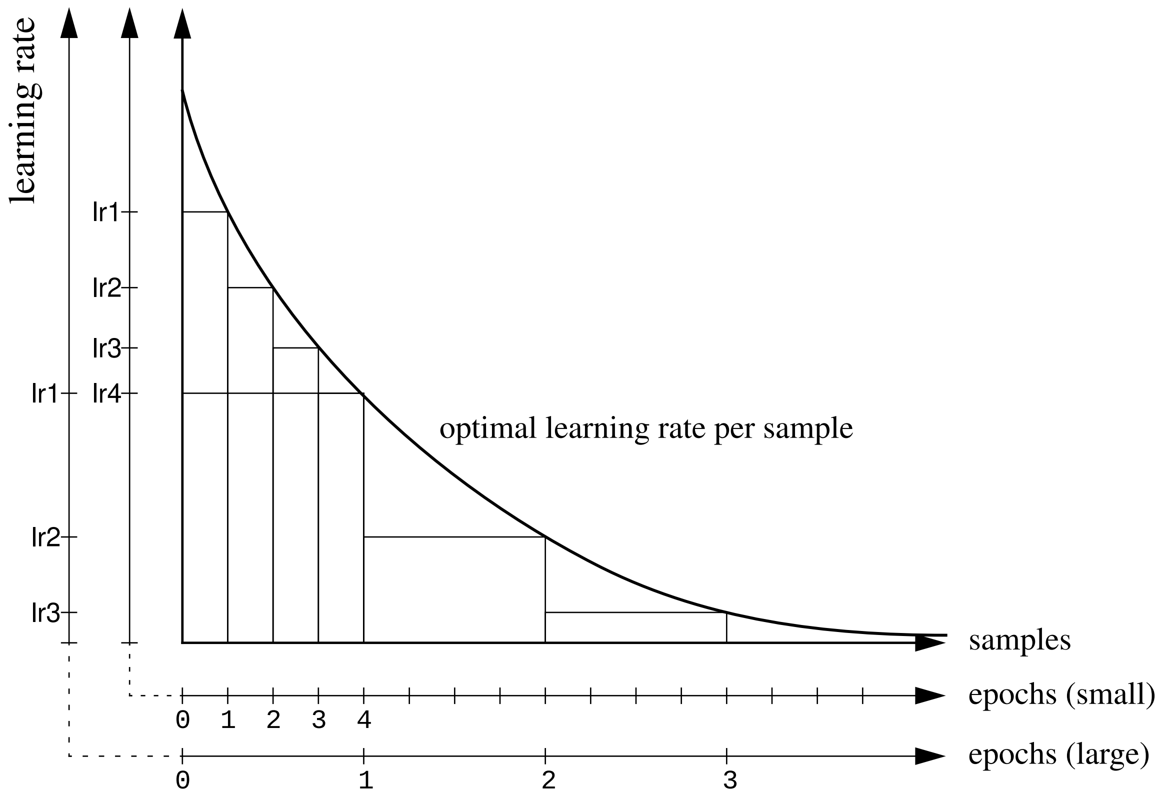


Figure 7.20: A larger training set should use smaller learning rates per epoch.

- **Normalization of inputs.** A greater standard deviation in the inputs implies larger learning rates, to compensate for the fact that the hidden units become saturated so the derivative of the sigmoid vanishes so learning is inhibited. (Unfortunately, these larger learning rates also lead to network oscillation, as we saw in Figure 7.12.)

- **Transfer functions in output layer.** Softmax outputs can initially use a larger learning rate than tanh outputs, apparently because softmax is "safer" than tanh, in the sense that the resulting output activations form better estimates of posterior probabilities during early training than tanh outputs do, since they are guaranteed to sum to 1.

- **Number of units.** It appears that more input units, regardless of their representation, imply smaller learning rates; and more hidden units imply a slower decay rate. Overall, it appears that the learning rate schedule becomes gentler as the network increases in size and its statistical fluctuations are smoothed out.

We found that the optimal learning rate schedule was less affected by other factors, such as the input representation, the use of shortcut connections, the hierarchy of delays, the speaker population (male vs. everyone), or the labels used during training. It remains unclear whether the schedule is affected by other factors, such as the weight update frequency or the use of different error criteria.

7.3.4.2. Momentum and Other Heuristics

Momentum is often a useful technique for hastening convergence in a neural network. Because it pushes weights further in a previously useful direction, momentum is most effective when the direction of change is fairly stable from one update to the next, implying that weights should not be updated after each training sample, but after a large number of training samples. Unfortunately, while momentum may increase the speed of convergence, it also destabilizes the learning rate schedule, and so it can be tricky to use. We tried using momentum in a few early experiments, in which training samples (frames) were presented in sequential order within a sentence rather than randomized order, and weights were updated after each sentence (~300 frames) rather than after each frame. We found that a momentum value of 0.9, which is often optimal in other domains, was too high and seriously degraded performance. A smaller value of 0.5 seemed somewhat helpful during the first iteration, but made no difference in subsequent iterations. We shortly thereafter abandoned the use of momentum, not wishing to complicate our system with a marginally useful technique.

Another technique that is often used to increase the convergence rate is a *derivative offset*. This is a value (typically 0.1) that is added to the derivative of the sigmoid function (one of the multiplicative factors in backpropagation), so that learning does not stall for saturated units whose sigmoid derivative is near zero. We performed some early experiments with a sigmoid derivative of 0.1, but we found it to be unnecessary for our data, so we soon stopped using it.

Networks that are trained as classifiers sometimes get stuck in a suboptimal corner of weight space, because it is difficult to learn the binary targets 0.0 and 1.0 (which lie in the saturation regions of a sigmoid) unless the network develops huge, dangerous weights. Many researchers avoid this problem by introducing a *target offset*, redefining the targets as 0.1 and 0.9 (well within the active region of the sigmoid), so the network can learn to classify the data using smaller, safer weights. We tried using target offsets for a while, but eventually realized that it undermined the ability of our system to estimate posterior probabilities. For example, after training had asymptoted, our 61 output units summed to something closer to $(60(0.1) + 1(0.9)) = 6.9$ than to $(60(0.0) + 1(1.0)) = 1.0$, so our outputs didn't resemble probabilities of any kind, and we were unable to take advantage of the probabilistic framework of HMMs. We concluded that target offsets are useful only in domains whose class distributions have virtually no overlap, such that the posterior probabilities that will be estimated by the network's outputs are virtually binary, subjecting the network to the problems of saturation. In the case of speech, class distributions overlap considerably, so target offsets are unnecessary, and even harmful.

7.3.4.3. Training Schedule

A training schedule has many dimensions; among these are the sequence of presentation of the training samples, and the weight update frequency. We will treat these two issues together, because we studied them together rather than independently.

Training samples may be presented in linear sequence (as they occur naturally) or in randomized sequence, where randomization may occur at the level of frames, sentences, and/or speakers. A totally randomized sequence is preferable because it exposes the network to the greatest diversity of data in any period of time, so the network is less liable to forget what it previously learned about any region of acoustic space.

Meanwhile, weights may be updated after every N training samples, for any value of N between 1 and N_{max}, i.e., the number of samples in the whole training set. (The case of N=1 is often called *online* training, while N>1 is called *batch* training.) Smaller values of N imply more randomness in the weights' trajectory, which means there is a lot of wasted movement (although this also helps the network escape from local minima); larger values of N imply a smoother trajectory, but longer intervals between weight updates. Fastest learning often results from using $N << N_{max}$, especially when training on a large database of fairly redundant data, as in the speech domain.

Our experiments did not cleanly separate these two dimensions of the training schedule, but instead considered them together. We worked with two kinds of training schedules, based on frames or sentences. In *frame-based* training, we presented frames in random order, and updated the weights after each frame. In *sentence-based* training, we presented frames in sequential order within a sentence, and updated the weights at the end of each sentence. Note that in both cases, we have $N << N_{max}$ = 1.2 million frames; but the former case uses online training, while the latter uses batch training (with $N \approx 300$).

In an early experiment, using sentence-based training (with geometric learning rates), we measured the benefit of randomizing the training sentences. The 3600 training sentences, representing an average of 36 sentences from each of 100 speakers, were ordered either serially (36 at a time from each speaker) or randomly (out of the 3600 sentences). We found that randomized sequencing reduced the error rate by about 15% during each iteration, asymptoting at 82% word accuracy vs. 79% in the serial case. We conclude that it is important to randomize the training sentences, because grouping them by speaker allows the network to focus for too long on the acoustic characteristics of the current speaker, eroding performance on the other speakers.

In later experiments, we found frame-based training to be significantly better than sentence-based training. In a direct comparison between these two approaches, using 3600 training sentences (with separately optimized learning rate schedules), frame-based training gave about 30% fewer errors than sentence-based training in each iteration (e.g., 88% vs 82% word accuracy after 3 iterations). However, further experiments would be required to determine how much of this improvement was due to randomized vs. serial frame-level sequencing, and how much was due to online vs. batch updating. We note that it has yet to be established whether online updating (N=1) really gives faster learning than batch updating with a small value of N (e.g, 10, 100, or even 300).

7.3.4.4. Gender Dependence

Speech recognition is difficult because of overlapping distributions. This problem is exacerbated in a speaker-independent system, because everyone has different voice characteristics, so the phonetic class distributions are further spread out, increasing their overlap and confusability. Recognition accuracy can be improved if the overlapping distributions can be teased apart by some form of clustering. A simple and elegant way to do this in a speaker independent system is to cluster the data by the speaker's gender. In other words, we can train one system on male data, and another on female data; subsequently we can recognize speech from an unknown speaker by first classifying the speaker's gender, and then applying the appropriate gender-dependent recognizer. This approach is particularly appealing because males and females have substantially different voice characteristics, so they significantly worsen the overlap and they are very easy to distinguish. (For example, Konig and Morgan (1993) found that a simple neural network can identify the gender of an utterance with 98.3% accuracy.)

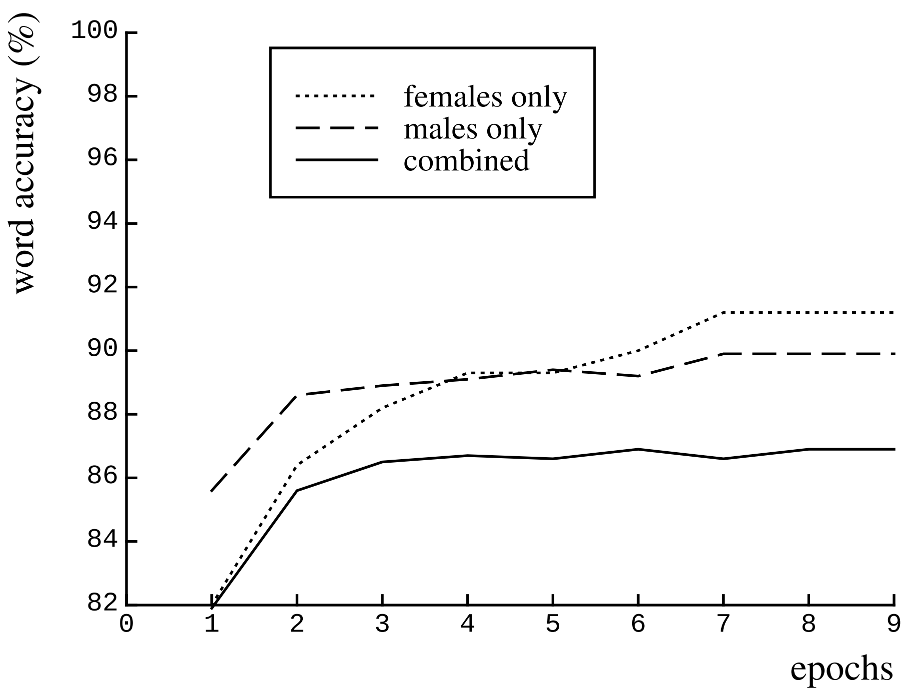


Figure 7.21: Gender dependent training improves results by separating two overlapping distributions.

Figure 7.21 shows the performance of three networks: a male-only network, a female-only network, and a mixed-gender network. The male network was trained on 2590 male sentences and tested on 240 male sentences; the female network was trained on 1060 female sentences and tested on 100 female sentences; and the mixed network was trained on 3600 mixed sentences and tested on 390 mixed sentences. We see that each of the gender dependent networks outperforms the mixed network, by a significant margin. The fact that the male and female networks outperformed the mixed network despite their relative poverty of training data testifies to the separability of male and female distributions.

We note that cross-gender testing gave poor results. For example, a network trained on male data achieved 89% word accuracy on male data, but only 61% on female data. For this reason, it may not even be necessary to identify the gender of the speaker with a separate network; it may work just as well to present the unknown utterance to both the male and female network, and to return the result from the system that obtained the highest DTW score. However, we did not have time to confirm the merit of this approach.

7.3.4.5. Recursive Labeling

In order to train a classifier network, we require a phonetic label (target class) for each frame in the database. These labels can be generated by any speech recognizer, by performing a Viterbi alignment between each utterance and its known phonetic pronunciation, thus identifying the correspondence between frames and states. The quality of the labels that are provided will affect the resulting word accuracy, i.e., high-quality labels will give better results than sloppy labels. As a system learns, it becomes capable of producing better and better labels itself; indeed, at some point it may even become capable of producing better labels than the ones it was trained on. When that happens, the system may be further improved by training it on these recursive labels instead of the original labels. This cycle can be repeated to the point of final optimality.

Our networks were initially trained on phonetic labels generated by SRI's DECIPHER system, provided to us through ICSI. (We note that DECIPHER achieved 86% word accuracy with the context-independent phone models from which these labels were generated.) We used these labels to train a network, and then we generated our own "second generation" labels via Viterbi alignment of the training data. We then used these second generation labels to train a gender-dependent network, and generated "third generation" labels. Figure 7.22 shows the performance of a gender-dependent network that was subsequently trained on each of these three generations of labels, under otherwise identical conditions (geometric learning rate schedule, male speakers). We see that each generation of labels improved the word accuracy somewhat, so the third generation resulted in 5-10% fewer errors than the first generation. We conclude that recursive labeling is another valuable technique for enhancing word accuracy.

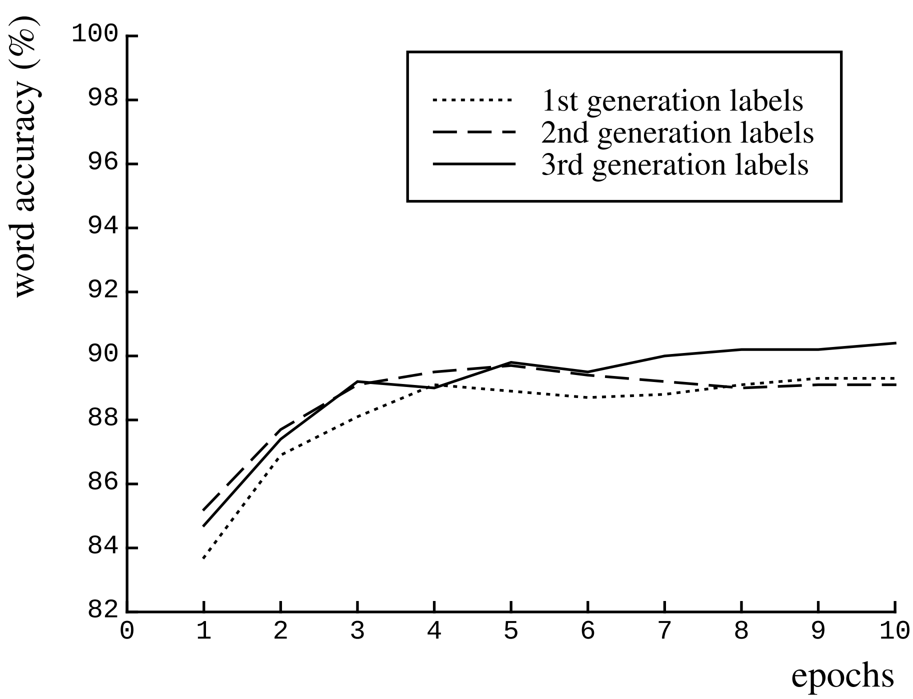


Figure 7.22: Recursive labeling optimizes the targets, and so improves accuracy.

7.3.5. Testing Procedures

A neural network, no matter how well it has been trained, will yield poor results unless it is used properly during testing. In this section we discuss the effectiveness of different testing procedures for our system.

7.3.5.1. Transforming the Output Activations

One might suggest at least three plausible ways to use the output activations of a classifier network to perform continuous speech recognition:

1. Apply DTW directly to these activations (scoring hypotheses by summing the activations along the alignment path). This approach might be prompted by a visual inspection of the output activations, noting that the network generally shows a high activation for the correct phoneme at each frame, and low activation for all incorrect phonemes.
2. Apply DTW to the *logarithms* of the activations (summing the log activations along the alignment path). This approach is motivated by the fact that the activations are estimates of probabilities, which should be multiplied rather than added, implying that their logarithms should be added.
3. Apply DTW to log ($Y_i/P(i)$), i.e., divide the activations by the priors (summing the log quotients along the alignment path). This approach is motivated by the fact that the activations are estimates of posterior probabilities. Recall that in an HMM, emission probabilities are defined as likelihoods $P(x|c)$, not posteriors $P(c|x)$; therefore, in an NN-HMM hybrid, during recognition, the posteriors should first be converted to likelihoods using Bayes Rule:

   $$P(x|c) = \frac{P(c|x) \cdot P(x)}{P(c)}$$

   where $P(x)$ can be ignored during recognition because it's a constant for all states in any given frame, so the posteriors $P(c|x)$ may be simply divided by the priors $P(c)$.

Each of these successive approaches is better justified than the previous ones; therefore we would expect the last approach to give the best results. This was confirmed by a direct comparison, which gave the following results:

DTW value	Word Accuracy
Y_i	74.9%
$\log Y_i$	90.6%
$\log(Y_i/P(i))$	91.5%

Table 7.1: Performance improves when output activations are transformed properly.

7.3.5.2. Duration Constraints

In a standard HMM, the state transition probabilities a_{ij} are reestimated during training, and these probabilities influence the duration of each state during recognition. Unfortunately, as we saw earlier, a self-transition with a constant probability implies an exponentially decaying duration model, rather than a more accurate bell-shaped model; moreover, it is well known that duration modeling plays a relatively small role in recognition accuracy. Therefore, in our NN-HMM hybrid, we chose to ignore the issue of reestimation of a_{ij}, and we simply assumed a uniform probability distribution for all a_{ij}. Meanwhile, we explored other types of duration constraints, i.e., hard minimum and maximum duration constraints at both the phoneme and word level, and probabilistic duration constraints applied to segments rather than frames.

In all cases, the durational statistics were obtained from the labeled training data. However, minimum phoneme durations taken from the training data proved not to be very helpful, since there is always some instance of each phoneme that is labeled with an essentially zero duration, rendering that minimum duration constraint useless during testing. Therefore we assigned a minimum duration to each phoneme equal to ξ times the phoneme's average duration in the training set; we obtained best results with $\xi = 0.5$ (searching at intervals of 0.1), so we used this value in all of our experiments.

7.3.5.2.1. Phoneme Duration Constraints

We first studied the effect of hard minimum and maximum phoneme duration constraints. There are at least two ways to impose such constraints:

1. Enforce constraints dynamically. That is, during DTW, keep track of the current duration of each state in each frame (implicit in the backtrace information), and place special restrictions on the final state of each phoneme, forcing it to self-transition until the phoneme has met its minimum duration requirement, or forcing it to exit-transition when the phoneme has met its maximum duration requirement.

2. Duplicate the states, and impose a pattern of transitions that enforce the duration constraints, as illustrated in Figure 7.23. Panel (a) shows how to use state duplication to enforce minimum duration constraints only; panel (b) shows how to enforce both minimum and maximum duration constraints.

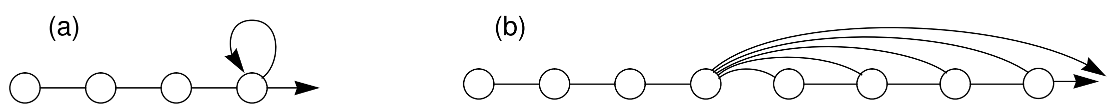


Figure 7.23: Duration constraints via state duplication. (a) 4 or more frames. (b) 4 to 8 frames.

Of these two approaches, state duplication clearly requires more memory, but it has the advantage that it gives correct results while the other method does not. The suboptimality of dynamically enforced constraints is demonstrated by the following examples.

Suppose we have a word with 3 states, and a minimum duration constraint of 2 frames per state. Figure 7.24 shows how this would be modeled using (a) dynamically enforced constraints and (b) state duplication. The solid line shows the only legal path through a 6-frame matrix. If, at the circled point in Figure 7.24(a), the diagonal predecessor is better than the horizontal predecessor, so that the dotted path is established to that point, then it will later be impossible to recover from that local decision, and the entire word will be rejected. By contrast, as shown in Figure 7.24(b), state duplication allows a diagonal path to proceed straight through this word with some cumulative score, so the word will never be rejected outright. Thus, state duplication is a safer strategy for implementing minimum duration constraints.

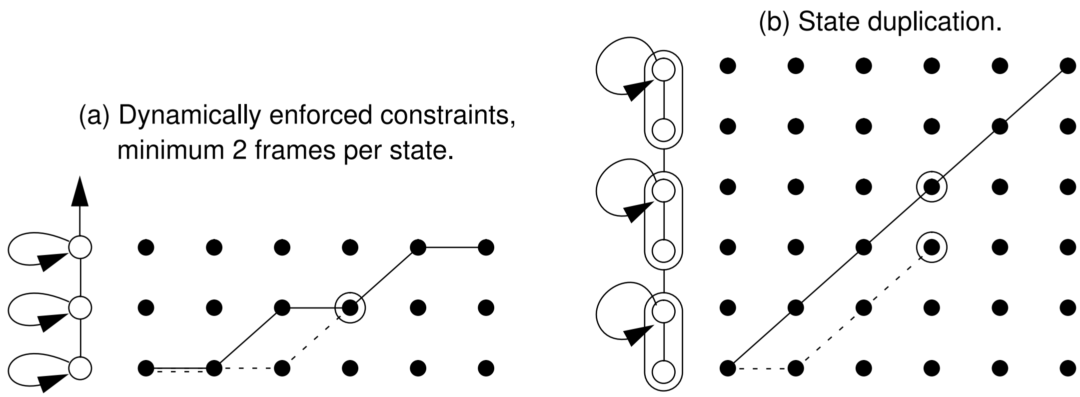


Figure 7.24: Minimum phoneme duration constraints. Only state duplication gives optimal results.

Experimentally, we found that minimum duration constraints were extremely helpful if implemented by state duplication, but actually harmful if they were enforced dynamically. For example, in a baseline experiment, when training on 3600 sentences and testing on 390 sentences, we obtained 74.5% word accuracy without duration constraints. When we imposed minimum duration constraints using state duplication, word accuracy jumped to 86.0%; but when we dynamically enforced the minimum duration constraints, accuracy degraded to 71.7%, apparently because too many words were being prematurely rejected on the basis of local decisions.

Maximum duration constraints are likewise more safely implemented with state duplication, as shown by the following example. Suppose we have a word with 3 states, and a maximum duration constraint of 4 frames per state. Figure 7.25 shows how this would be modeled using (a) dynamically enforced constraints and (b) state duplications. In Figure 7.25(a), if point *a* has a local score of 1 while point *b* has a local score of 2, then point *c* will choose *b* as its predecessor, establishing the path along the solid line. However, this local decision, combined with the maximum duration constraint of 4 frames, will prevent the path from reaching point *d* which has a local score of 3, and instead we will have to settle for point *e* which has a local score of only 1, so the cumulative score along the solid line will be worse than if the transition into the middle state had been delayed by one frame. By contrast, Figure 7.25(b) shows that state duplication permits entry into the middle state to be postponed, so we can determine the true optimal path (dashed line).

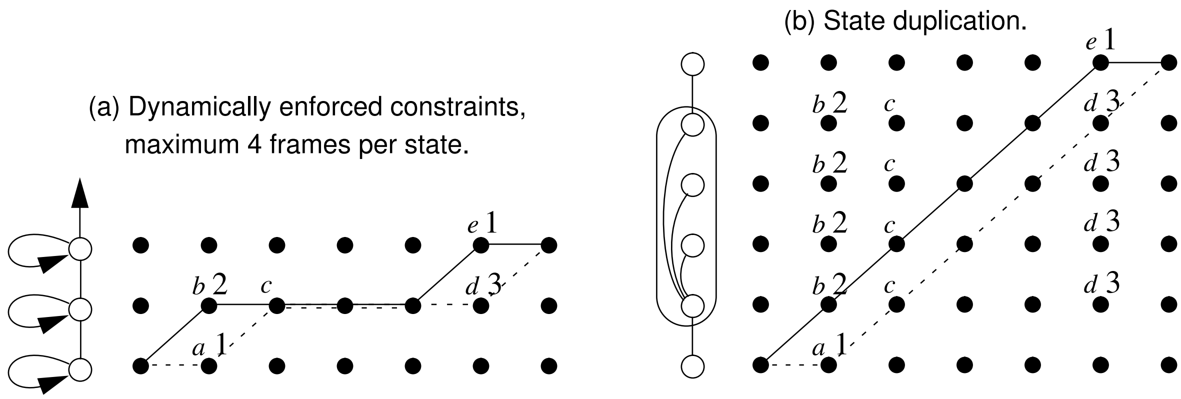


Figure 7.25: Maximum phoneme duration constraints. Only state duplication gives optimal results.

Because the phonemes in our hypotheses tend to be too short rather than too long, we did not expect maximum phoneme duration constraints to make much difference. Indeed, we found experimentally that enforcing maximum duration constraints dynamically had no effect on word accuracy. Due to a lack of time we did not investigate state duplication, but we do not expect that it would make much difference either.

We also tried phoneme duration constraints based on segment duration probabilities, instead of hard minimum and maximum limits. In this approach, we used the labeled training data to construct a histogram of durations for each phoneme (up to 25 frames), as illustrated in Figure 7.26 for some typical phonemes. Then, during recognition, whenever transitioning out of the final state of any phoneme, we added a penalty of $\kappa \cdot \log P(dur)$,

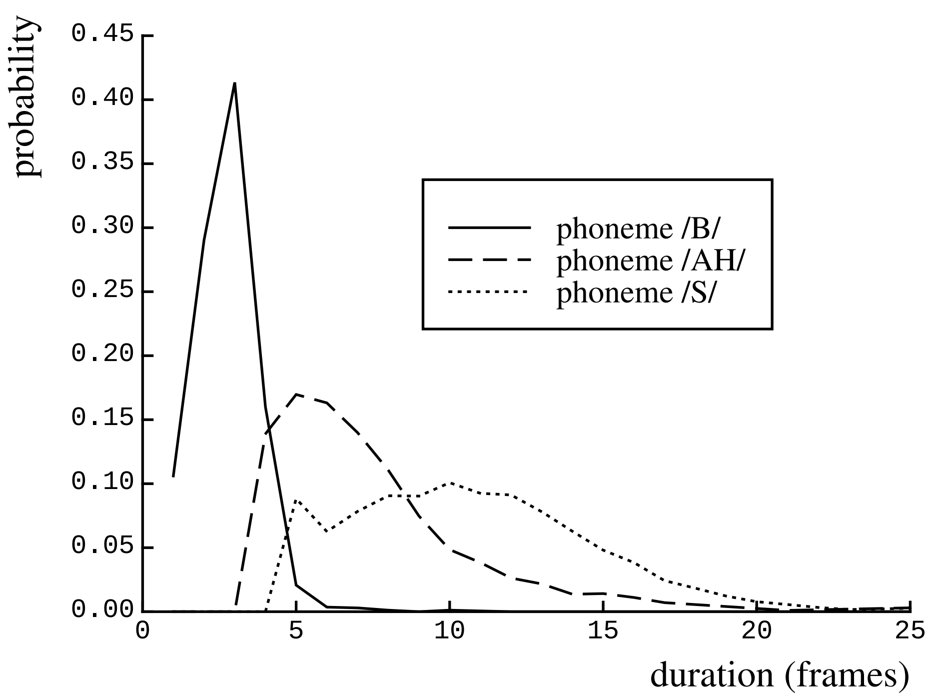


Figure 7.26: Histogram of durations of some typical phonemes.

where *dur* is the segment duration of that phoneme (implicit in the backtrace), $P(dur)$ is the probability of that duration for that phoneme (according to the appropriate histogram), and κ is a scaling factor. Experiments showed that $\kappa = 1$ was the best scaling factor, and that the best word accuracy we could achieve with this approach was 85.4%. We concluded that hard minimum and maximum limits are more effective than a probabilistic model, so we reverted to that approach in our subsequent experiments.

7.3.5.2.2. Word Duration Constraints

We tried extending hard duration constraints to the word level, using both state duplication and dynamic enforcement, with only limited success. At the word level, there is no longer any guarantee that state duplication will give optimal results, because we must make an arbitrary decision about how to distribute the duplication of states over all the phonemes of the word, and this distribution may be suboptimal. In our experiments with state duplication, we tried distributing the states evenly over all the phonemes (or evenly distributing the "leftover" states if minimum phoneme duration constraints were also used). We found that this gave worse results than using no contraints at all, i.e., word accuracy degraded from 74.5% to 70.6%, or from 86.0% to 84.3% if minimum phoneme duration constraints were also being used. This degradation probably reflected the fact that states should be distributed not evenly, but in proportion to their average phoneme duration, or in some other statistical fashion; but we did not have time to investigate this further.

We then studied word duration constraints using dynamic enforcement, i.e., keeping track of the current duration of each word in each frame (implicit in the backtrace from the final state of the word), and requiring the final state to self-loop until it met the word's minimum duration requirement. This improved the word accuracy from 74.5% to 79.8%, or even to 85.7% when combined with phoneme duration constraints. Note, however, that this final result was a degradation from 86.0% when using only phoneme duration constraints. We further extended this approach by dynamically enforcing up to three minimum durations for each word, corresponding to three different amounts of word context, i.e., the word by itself, vs. the word preceded by any other word, vs. the word preceded by any other two words. This was motivated by our observation that some hypotheses contained strings of adjacent words all having impossibly short durations. Unfortunately, each of these additional constraints further degraded word accuracy, from 86.0% without word constraints, to 85.7%, 85.3%, and 85.0% respectively (and cumulatively) using single, double, and triple minimum word duration constraints. Maximum word durations also degraded performance, for example from 85.0% to 81.5% accuracy.

We believe that word duration constraints were dangerous largely because they were based on insufficient statistics. All of our duration constraints were derived from the 3600 training sentences. While this represents an average of 3000 instances of each of our 61 phonemes, it represents an average of only 30 samples of each of the 1000 words in our vocabulary, which is a dangerously small population. We tried to compensate for this fact by relaxing the constraints, e.g., shaving 30% off the minimum word durations and adding 30% to the maximum word durations seen in the training set; but this gave no significant improvement. We conclude that hard word duration constraints are not reliable enough to be useful.

7.3.5.3. Word Transition Penalties

As in a standard HMM, we found it useful to balance the acoustic modeling against the language modeling by using word transition penalties, i.e., adding a constant penalty during every transition out of any word. Values of -15 to -20 generally gave best results, when we performed Viterbi search on $\log(Y_i/P(i))$. (Values of -2 to -4 were better when we performed Viterbi search on Y_i directly.)

7.3.6. Generalization

We conclude our discussion of frame level training by presenting some results on generalization. Of course performance will always be better on the training set than on the test set. If a system were trained on just a few sentences, the system could learn to memorize the patterns and score 100% on the training set; but it would fail miserably on the independent test set, because the testing data would be too different from the training data. With more training sentences, the network will lose the ability to memorize all of the training data, but because the system is exposed to a more representative range of data, its performance on the test set will rapidly improve. With still more training data, the system will form an increasingly accurate model of the distribution in acoustic space, and performance will steadily improve on both the training and testing sets. This is the reason behind the adage, "There's no data like more data."

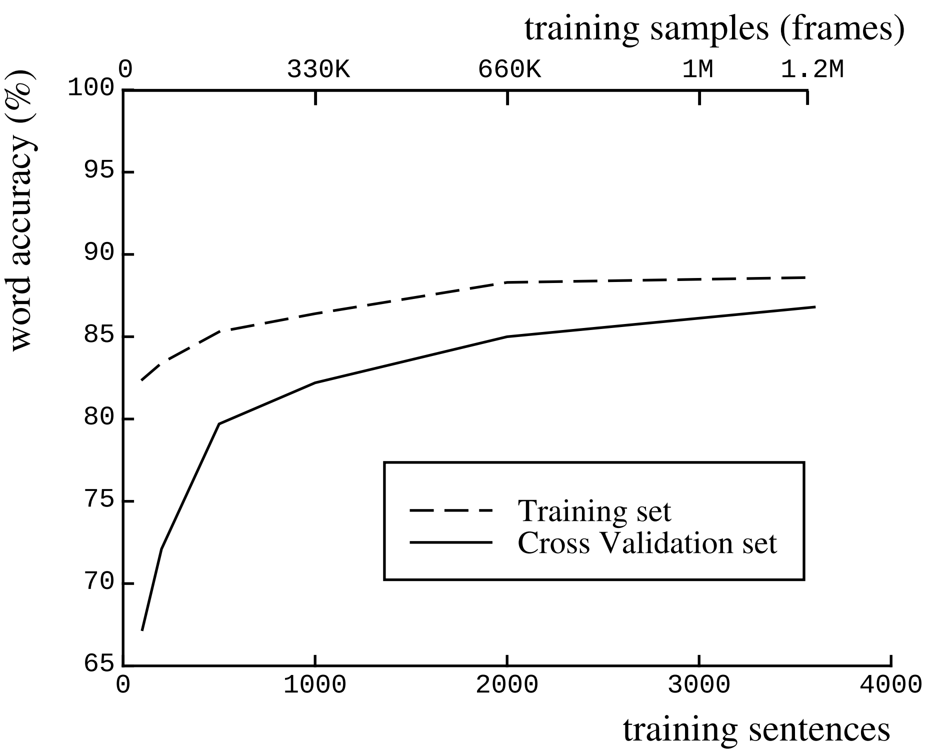


Figure 7.27: Generalization improves with more training data. The training-testing gap also shrinks.

We measured the asymptotic word accuracy on both the training set and a cross validation set, using a system with 100 hidden units (i.e., 21,000 weights) that was trained on either 100, 200, 500, 1000, 2000, or 3600 sentences; results are shown in Figure 7.27. As expected, we see that performance steadily improves on both the training and cross validation sets, given increasing amounts of training data. The immediate rise in accuracy on the training set implies that 100 training sentences (33,000 frames) is already too much for the network to memorize; thus, all of the improvements in this graph arise from more accurate modeling of the distribution in acoustic space.

7.4. Word Level Training

All of the experiments described so far used frame level training, i.e., outputs and targets that are defined on a frame-by-frame basis. We have seen how to optimize the performance of such a system, by exploiting the fact that its outputs provide estimates of posterior probabilities, and using techniques such as division by priors, expanded window sizes, optimized learning rate schedules, gender dependent training, and duration constraints. While each of these optimizations leads to better performance, there is a natural limit associated with the use of frame level training.

One fundamental problem with frame level training is that the training criterion is inconsistent with the testing criterion — that is, the training criterion is framewise phoneme classification accuracy, while the testing criterion is word recognition accuracy. We saw in Section 6.3.5 (in the context of predictive networks) that there may be only a weak correlation between phoneme accuracy and word accuracy, therefore we can expect better performance from a system that consistently uses word accuracy as both the training and testing criterion.

In order to perform *word level training*, we must define a neural network that classifies a whole word at a time (i.e., its inputs represent all the frames of speech in a whole word, and its outputs represent the *N* words in the vocabulary), so that we can compare the output activations against the desired targets of "1" for the correct word and "0" for all incorrect words, and backpropagate error through the whole network. Such a network must accept a variable number of input frames, therefore it should be a *dynamic* network (i.e., it should integrate local evidence over the duration of a word), as in a TDNN; meanwhile it should also use shared subword units (like phonemes) in order to scale well, thus it should be a *state-based* network, as in an HMM.

7.4.1. Multi-State Time Delay Neural Network

An interesting network that combines the above two features is the *Multi-State Time Delay Neural Network* (Haffner and Waibel, 1992). As can be seen in Figure 7.1, the MS-TDNN is an extension of the TDNN from the phoneme level to the word level, and from a single state to multiple states. That is, while a TDNN performs temporal integration by summing the activations of a single phoneme (a single state) over the duration of the phoneme, by contrast an MS-TDNN performs temporal integration by applying DTW to a sequence of states (comprising a word) over the duration of a word.

We will see in the next section that we obtained better word accuracy with word-level training than with frame-level training. But first we must describe the MS-TDNN in greater detail, presenting and motivating the details of its design. We will take an incremental approach, stepping through a series of possible designs, and showing how each improves on the earlier designs by resolving subtle inconsistencies, leading up to the design of the MS-TDNN that we actually used in our experiments.

Figure 7.28(a) shows a baseline system (with frame-level training), i.e., a simple TDNN whose phoneme outputs are copied into a DTW matrix, in which continuous speech is performed. As already noted, this system is suboptimal because the training criterion is inconsistent with the testing criterion: phoneme classification is not word classification.

To address this inconsistency, as argued above, we must train the network explicitly to perform word classification. To this end, we shall define a word layer with one unit for each word in the vocabulary, as illustrated in Figure 7.28(b) for the particular word “cat”. We correlate the activation of the word unit with the associated DTW score by establishing connections from the DTW alignment path to the word unit. Also, we give the phonemes within a word independently trainable weights, to enhance word discrimination (for example, to discriminate “cat” from “mat” it may be useful to give special emphasis to the first phoneme); these weights are tied over all frames in which the phoneme occurs. Thus a word unit is an ordinary unit, except that its connectivity to the preceding layer is determined dynamically, and its net input should be normalized by the total duration of the word. The word unit is trained on a target of 1 or 0, depending if the word is correct or incorrect for the current segment of speech, and the resulting error is backpropagated through the entire network. Thus, word discrimination is treated very much like phoneme discrimination.

Although network (b) resolves the original inconsistency, it now suffers from a secondary one — namely, that the weights leading to a word unit are used during training but ignored during testing, since DTW is still performed entirely in the DTW layer. We resolve this inconsistency by “pushing down” these weights one level, as shown in Figure 7.28(c). Now the phoneme activations are no longer directly copied into the DTW layer, but instead are modulated by a weight and bias before being stored there (DTW units are linear); and the word unit has constant weights, and no bias. During word level training, error is still backpropagated from targets at the word level, but biases and weights are modified only at the DTW level and below. Note that this transformed network is not exactly equivalent to the previous one, but it preserves the properties that there are separate learned weights associated with each phoneme, and there is an effective bias for each word.

Network (c) is still flawed by a minor inconsistency, arising from its sigmoidal word unit. The problem does not exist for isolated word recognition, since any monotonic function (sigmoidal or otherwise) will correlate the highest word activation with the highest DTW score. However, for continuous speech recognition, which concatenates words into a sequence, the optimal sum of sigmoids may not correspond to the optimal sigmoid of a sum, leading to an inconsistency between word and sentence recognition. Linear word units, as

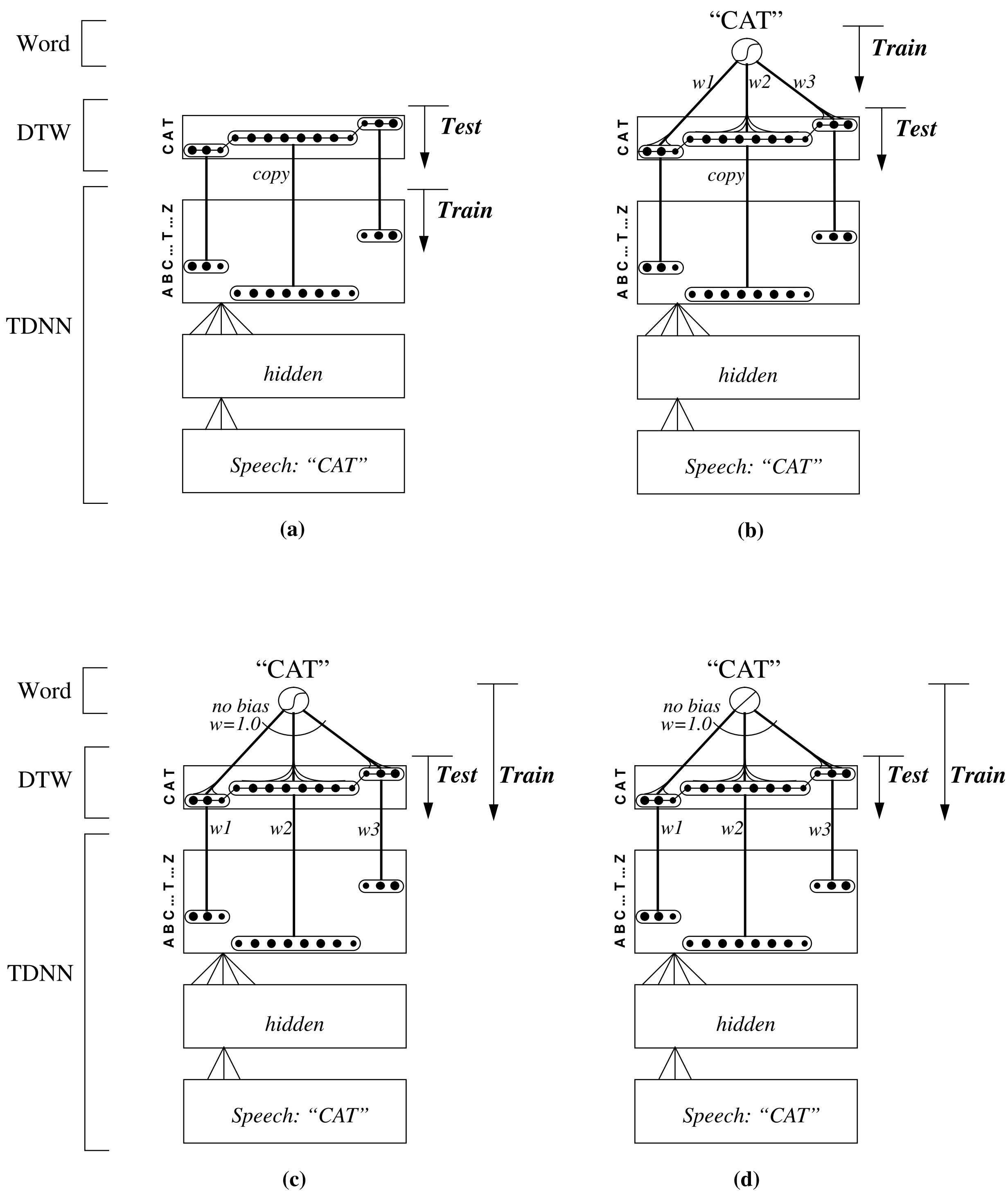


Figure 7.28: MS-TDNN, designed by resolving inconsistencies. (a) TDNN+DTW. (b) Adding word layer. (c) Pushing down weights. (d) Linear word units, for continuous speech recognition.

shown in Figure 7.28(d), would resolve this problem; in practice we have found that linear word units perform slightly better than sigmoidal word units.

At least two potential inconsistencies remain in network (d). First, the MS-TDNN training algorithm assumes that the network connectivity is fixed; but in fact the connectivity at the word level varies, depending on the DTW alignment path during the current iteration. Of course, as the training asymptotes and the segmentation stabilizes, this becomes a negligible issue. A more serious inconsistency can arise if discriminative training is performed at known word boundaries, because the word boundaries are in fact unknown during testing. This inconsistency could be resolved by discriminating against words within boundaries that are found by a free alignment during training, as suggested by Hild (1993). Unfortunately, this is an expensive operation, and it proved impractical for our system. Therefore, we settled on network (d), with known word boundaries during training, for our word level experiments.

The MS-TDNN has a fairly compact design. Note that its first three layers (the TDNN) are shared by all words in the vocabulary, while each word requires only one non-shared weight and bias for each of its phonemes. Thus the number of parameters remains moderate even for a large vocabulary, and the system can cope with limited training data. Moreover, new words can be added to the vocabulary without retraining, by simply defining a new DTW layer for each new word, with incoming weights and biases initialized to 1.0 and 0.0, respectively.

Given constant weights under the word layer, it may be argued that word level training is really just another way of viewing DTW level training; but the former is conceptually simpler because there is a single binary target for each word, which makes word level discrimination very straightforward. For a large vocabulary, discriminating against all incorrect words would be very expensive, so we discriminate against only a small number of close matches (typically one).

7.4.2. Experimental Results

We evaluated the MS-TDNN on both the Conference Registration database and the Resource Management database. These two sets of experiments were performed under rather different experimental conditions, both because these databases are of different sizes, and also because there was a delay of two years between these experiments, during which time we developed a better approach to frame-level training with techniques that carried over to our word-level experiments. We begin this section by summarizing the experimental conditions for each database.

In the Conference Registration experiments, we used an MS-TDNN with 16 melscale spectral coefficients (with 3 time delays), 20 hidden units (with 5 time delays), 120 phoneme units (40 phonemes with 3 states each), 5487 DTW units, and 402 word units, giving a total of 24,074 weights. The network used symmetric [-1,1] unit activations and inputs, and linear DTW units and word units. The system was first bootstrapped to asymptotic performance using frame level training. Word level training was then performed using the Classification Figure of Merit (CFM) error function, $E = (1 + (Y_{\bar{c}} - Y_c))^2$, in which the correct word (with activation Y_c) is explicitly discriminated from the best incorrect word (with activation $Y_{\bar{c}}$) (Hampshire and Waibel 1990a). CFM proved somewhat better than MSE for word level training, although the opposite was true for frame level training. Negative word level training was performed only if the two words were sufficiently confusable (i.e., if $Y_c - Y_{\bar{c}} < 0.3$), in order to avoid disrupting the network on behalf of words that had already been well-learned.

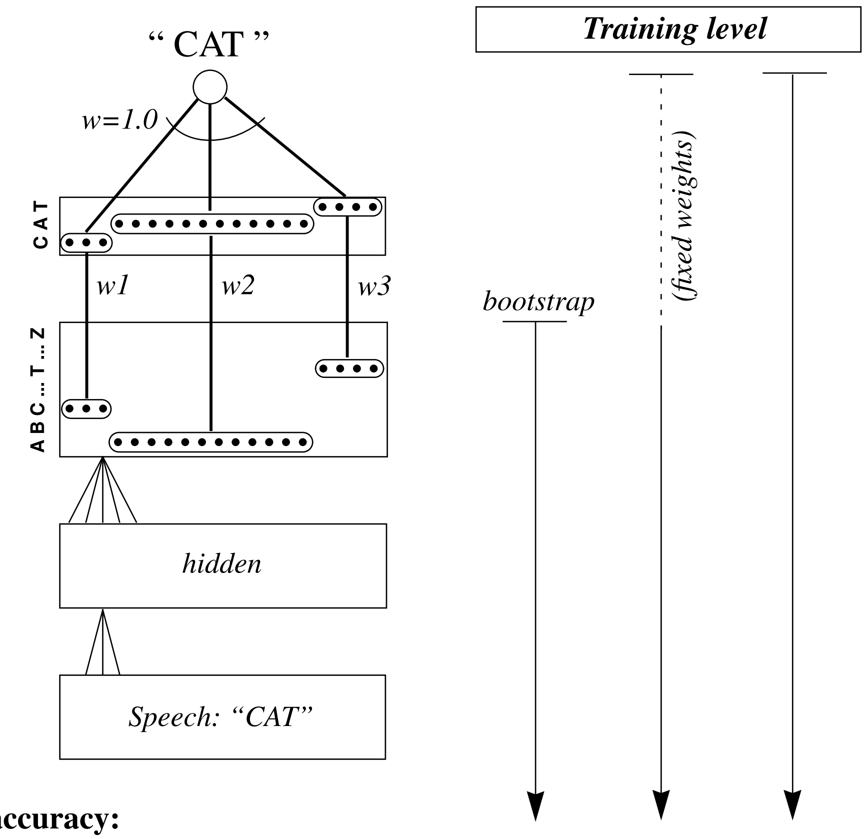
In our Resource Management experiments, we used an MS-TDNN with 16 LDA coefficients (with 9 time delays), 100 hidden units, 61 phoneme units, 6429 DTW units, and 994 word units, giving a total of 33,519 weights. The hidden layer used tanh activations, and the phoneme layer used softmax activations (preserving the MLP's bootstrap conditions); but the DTW units and word units were still linear. By this time, we understood that frame level training (used during the bootstrapping phase) yields phoneme activations that estimate the posterior probabilities, so we computed the net input to a DTW unit by

$$X = \text{bias} + \text{weight} \cdot \log\left(\frac{Y_i}{(P(i))}\right) \qquad \text{rather than} \qquad X = \text{bias} + \text{weight} \cdot Y_i$$

where Y_i is the activation of the corresponding phoneme unit, and $P(i)$ is its prior probability. Also, in these experiments, the learning rate schedule was optimized by dynamic search, rather than fixed at a constant value as in the Conference Registration experiments.

We found that different amounts of training were necessary for these two sets of experiments. In the Conference Registration experiments, we typically bootstrapped with frame level training for about 30 iterations, and then continued with word level training for another 10 iterations. For the Resource Management experiments, on the other hand, we typically bootstrapped with frame level training for only about 7 iterations, and then continued with word level training for another 2 iterations. The RM database required fewer iterations of training, both because it has 15 times as much training data, and also because our training techniques had improved in the two years between these experiments.

Figure 7.29 shows that for both databases, word level training gave significantly better word accuracy than frame level training. For example, on the Conference Registration database, word accuracy was 72% after frame level training, and 81% after word level training (representing a 32% reduction in the error rate); and on the Resource Management database, word accuracy was 89.2% after frame level training, and 90.5% after word level training (representing a 12% reduction in the error rate). This improvement was partly due to the increase in the number of weights in the system (from 13K to 24K, or from 21K to 33K); but we determined that it was also partly due to the word level training itself. To show this, we performed an intermediate experiment on each database, in which we trained the network at the word level, but we updated only the weights below the phoneme layer (as during bootstrapping), keeping the DTW weights fixed; results were 75% and 89.9% word accuracy for the two databases. Thus, even without adding any new weights, word level training leads to better word accuracy.



word accuracy:

Conference Registration database:	**72%**	**75%**	**81%**
Resource Management database:	**89.2%**	**89.9%**	**90.5%**

Figure 7.29: Word level training is better than frame level training, even if no new weights are added.

7.5. Summary

In this chapter we have seen that good word recognition accuracy can be achieved using neural networks that have been trained as speech classifiers. However, the networks cannot be simply thrown at the problem; they must be used carefully, and optimized well.

Table 7.2 summarizes the most important optimizations we made to our system, ranked by their relative impact on performance. We note that these values were all derived under particular experimental conditions, and that the values will change under different conditions, because these factors are nonlinearly related to each other. Thus, this table represents only a rough ranking of the value of these techniques.

Technique	Word accuracy (before)	Word accuracy (after)	Error Reduction
10 $\rightarrow$ 400 hidden units	71.8%	90.9%	68%
DTW uses $Y_i \rightarrow \log(Y_i/(P(i)))$	74.9%	91.5%	66%
1 $\rightarrow$ 9 input frames	80.4%	89.6%	47%
state duplication	74.5%	86.0%	45%
normalized inputs	67.2%	77.9%	33%
batch $\rightarrow$ online weight updates	82.0%	88.0%	33%
asymmetric $\rightarrow$ symmetric sigmoids	72.9%	81.4%	31%
gender dependence	86.9%	90.6%	28%
constant $\rightarrow$ dynamic learning rates	88.2%	90.7%	21%
grouped $\rightarrow$ randomized sentences	79.0%	82.0%	14%
word level training	89.2%	90.5%	12%
1 $\rightarrow$ 3 states per phoneme	86.6%	88.2%	12%
recursive labeling	89.3%	90.4%	10%
FFT $\rightarrow$ LDA inputs	86.0%	86.6%	4%

Table 7.2: Ranking of techniques by their impact on performance.

In our experiments using the Resource Management database, we ordinarily trained on 3600 sentences and tested on a cross-validation set of 390 speaker-independent sentences. However, we periodically evaluated performance as well on 600 test sentences representing the combined February 1989 and October 1989 official test sets. Our results were generally somewhat worse on the official test set; for example, our best results were 91.9% on the cross validation set, but only 90.5% on the official test set.

Figure 7.30 shows the performance of several versions of our system on the official test set. There were a number of differences between successive versions of our system (including some factors that went back and forth as we kept experimenting), but the primary factors that changed were as follows:

1. Baseline system, already incorporating the most important techniques in Table 7.2.
2. Normalized PLP inputs; better learning rate schedule.
3. Online weight update; softmax outputs.
4. Search for optimal learning rate schedule; gender dependence; LDA inputs.
5. Word level training.

All of these techniques, used together, contributed to an official word accuracy of 90.5% (i.e., a word error rate of 9.5%) using context-independent phoneme models with only 67K parameters. The final version of our system is described in detail in Appendix A.

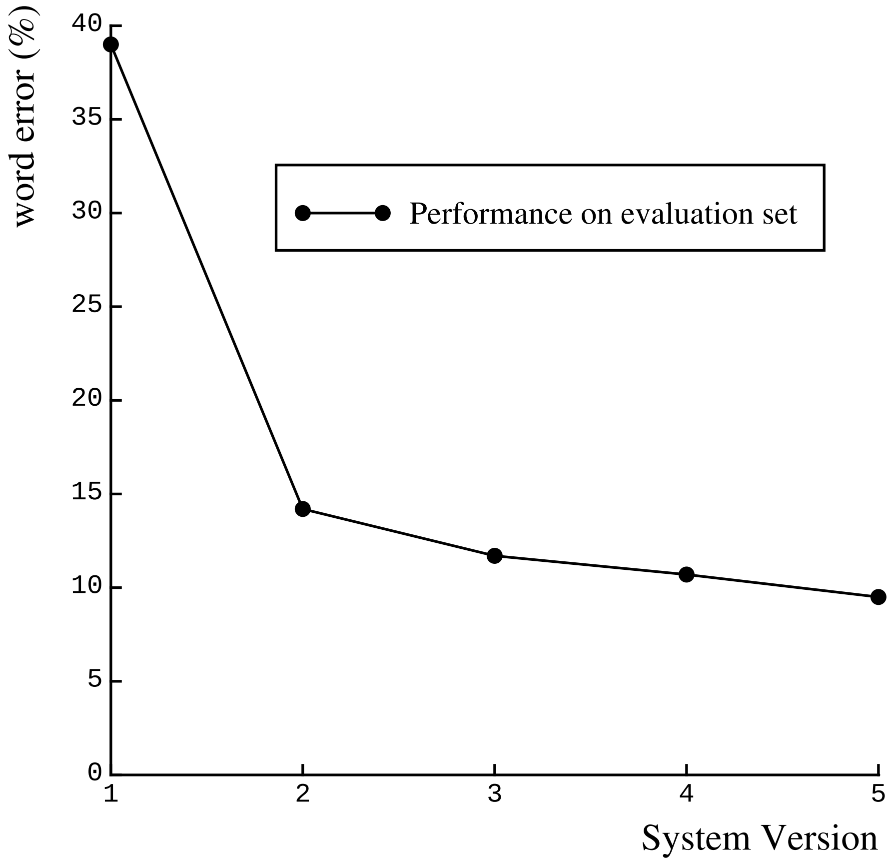


Figure 7.30: Snapshots of system performance on official evaluation set.

8. Comparisons

In this chapter we compare the performance of our best NN-HMM hybrids against that of various other systems, on both the Conference Registration database and the Resource Management database. These comparisons reveal the relative weakness of predictive networks, the relative strength of classification networks, and the importance of careful optimization in any given approach.

8.1. Conference Registration Database

Table 8.1 shows a comparison between several systems (all developed by our research group) on the Conference Registration database. All of these systems used 40 phoneme models, with between 1 and 5 states per phoneme. The systems are as follows:

- **HMM-*n***: Continuous density Hidden Markov Model with 1, 5, or 10 mixture densities per state (as described in Section 6.3.5).
- **LPNN**: Linked Predictive Neural Network (Section 6.3.4).
- **HCNN**: Hidden Control Neural Network (Section 6.4), augmented with context dependent inputs and function word models.
- **LVQ**: Learned Vector Quantization (Section 6.3.5), which trains a codebook of quantized vectors for a tied-mixture HMM.
- **TDNN:** Time Delay Neural Network (Section 3.3.1.1), but without temporal integration in the output layer. This may also be called an MLP (Section 7.3) with hierarchical delays.
- **MS-TDNN**: Multi-State TDNN, used for word classification (Section 7.4).

In each experiment, we trained on 204 recorded sentences from one speaker (mjmt), and tested word accuracy on another set (or subset) of 204 sentences by the same speaker. Perplexity 7 used a word pair grammar derived from and applied to all 204 sentences; perplexity 111 used no grammar but limited the vocabulary to the words found in the first three conversations (41 sentences), which were used for testing; perplexity 402(a) used no grammar with the full vocabulary and again tested only the first three conversations (41 sentences); perplexity 402(b) used no grammar and tested all 204 sentences. The final column gives the word accuracy on the training set, for comparison.

System	perplexity 7	111	402(a)	402(b)	test on training set 111
HMM-1		55%			
HMM-5	96%	71%	58%		76%
HMM-10	97%	75%	66%		82%
LPNN	97%	60%	41%		
HCNN		75%			
LVQ	98%	84%	74%	61%	83%
TDNN	98%	78%	72%	64%	
MS-TDNN	98%	82%	81%	70%	85%

Table 8.1: Comparative results on the Conference Registration database.

The table clearly shows that the LPNN is outperformed by all other systems except the most primitive HMM, suggesting that predictive networks suffer severely from their lack of discrimination. On the other hand, the HCNN (which is also based on predictive networks) achieved respectable results, suggesting that our LPNN may have been poorly optimized, despite all the work that we put into it, or else that the context dependent inputs (used only by the HCNN in this table) largely compensate for the lack of discrimination. In any case, neither the LPNN nor the HCNN performed as well as the discriminative approaches, i.e., LVQ, TDNN, and MS-TDNN.

Among the discriminative approaches, the LVQ and TDNN systems had comparable performance. This reinforces and extends to the word level McDermott and Katagiri's conclusion (1991) that there is no significant difference in phoneme classification accuracy between these two approaches — although LVQ is more computationally efficient during training, while the TDNN is more computationally efficient during testing.

The best performance was achieved by the MS-TDNN, which uses discriminative training at both the phoneme level (during bootstrapping) and at the word level (during subsequent training). The superiority of the MS-TDNN suggests that optimal performance depends not only on discriminative training, but also on tight consistency between the training and testing criteria.

8.2. Resource Management Database

Based on the above conclusions, we focused on discriminative training (classification networks) when we moved on to the speaker independent Resource Management database. Most of the network optimizations discussed in Chapter 7 were developed on this database, and were never applied to the Conference Registration database.

Table 8.2 compares the results of various systems on the Resource Management database, including our two best systems (in boldface) and those of several other researchers. All of these results were obtained with a word pair grammar, with perplexity 60. The systems in this table are as follows:

- **MLP**: our best multilayer perceptron using virtually all of the optimizations in Chapter 7, except for word level training. The details of this system are given in Appendix A.
- **MS-TDNN**: same as the above system, plus word level training.
- **MLP** (ICSI): An MLP developed by ICSI (Renals et al 1992), which is very similar to ours, except that it has more hidden units and fewer optimizations (discussed below).
- **CI-Sphinx**: A context-independent version of the original Sphinx system (Lee 1988), based on HMMs.
- **CI-Decipher**: A context-independent version of SRI's Decipher system (Renals et al 1992), also based on HMMs, but enhanced by cross-word modeling and multiple pronunciations per word.
- **Decipher**: The full context-dependent version of SRI's Decipher system (Renals et al 1992).
- **Sphinx-II**: The latest version of Sphinx (Hwang and Huang 1993), which includes senone modeling.

System	type	parameters	models	test set	word accuracy
MLP	**NN-HMM**	**41,000**	61	Feb89+Oct89	**89.2%**
MS-TDNN	**NN-HMM**	**67,000**	61	Feb89+Oct89	**90.5%**
MLP (ICSI)	NN-HMM	156,000	69	Feb89+Oct89	87.2%
CI-Sphinx	HMM	111,000	48	Mar88	84.4%
CI-Decipher	HMM	126,000	69	Feb89+Oct89	86.0%
Decipher	HMM	5,500,000	3,428	Feb89+Oct89	95.1%
Sphinx-II	HMM	9,217,000	7,549	Feb89+Oct89	96.2%

Table 8.2: Comparative results on the Resource Management database (perplexity 60).

The first five systems use context independent phoneme models, therefore they have relatively few parameters, and get only moderate word accuracy (84% to 91%). The last two systems use context dependent phoneme models, therefore they have millions of parameters, and they get much higher word accuracy (95% to 96%); these last two systems are included in this table only to illustrate that state-of-the-art performance requires many more parameters than were used in our study.

We see from this table that the NN-HMM hybrid systems (first three entries) consistently outperformed the pure HMM systems (CI-Sphinx and CI-Decipher), using a comparable number of parameters. This supports our claim that neural networks make more efficient use of parameters than an HMM, because they are naturally discriminative — that is, they model posterior probabilities $P(class|input)$ rather than likelihoods $P(input|class)$, and therefore they use their parameters to model the simple boundaries between distributions rather than the complex surfaces of distributions.

We also see that each of our two systems outperformed ICSI's MLP, despite ICSI's relative excess of parameters, because of all the optimizations we performed in our systems. The most important of the optimizations used in our systems, and not in ICSI's, are gender dependent training, a learning rate schedule optimized by search, and recursive labeling, as well as word level training in the case of our MS-TDNN.

Finally, we see once again that the best performance is given by the MS-TDNN, reconfirming the need for not only discriminative training, but also tight consistency between training and testing criteria. It is with the MS-TDNN that we achieved a word recognition accuracy of 90.5% using only 67K parameters, significantly outperforming the context independent HMM systems while requiring fewer parameters.

9. Conclusions

This dissertation has addressed the question of whether neural networks can serve as a useful foundation for a large vocabulary, speaker independent, continuous speech recognition system. We succeeded in showing that indeed they can, when the neural networks are used carefully and thoughtfully.

9.1. Neural Networks as Acoustic Models

A speech recognition system requires solutions to the problems of both acoustic modeling and temporal modeling. The prevailing speech recognition technology, Hidden Markov Models, offers solutions to both of these problems: acoustic modeling is provided by discrete, continuous, or semicontinuous density models; and temporal modeling is provided by states connected by transitions, arranged into a strict hierarchy of phonemes, words, and sentences.

While an HMM's solutions are effective, they suffer from a number of drawbacks. Specifically, the acoustic models suffer from quantization errors and/or poor parametric modeling assumptions; the standard Maximum Likelihood training criterion leads to poor discrimination between the acoustic models; the Independence Assumption makes it hard to exploit multiple input frames; and the First-Order Assumption makes it hard to model coarticulation and duration. Given that HMMs have so many drawbacks, it makes sense to consider alternative solutions.

Neural networks — well known for their ability to learn complex functions, generalize effectively, tolerate noise, and support parallelism — offer a promising alternative. However, while today's neural networks can readily be applied to static or temporally localized pattern recognition tasks, we do not yet clearly understand how to apply them to dynamic, temporally extended pattern recognition tasks. Therefore, in a speech recognition system, it currently makes sense to use neural networks for acoustic modeling, but not for temporal modeling. Based on these considerations, we have investigated hybrid NN-HMM systems, in which neural networks are responsible for acoustic modeling, and HMMs are responsible for temporal modeling.

9.2. Summary of Experiments

We explored two different ways to use neural networks for acoustic modeling. The first was a novel technique based on **prediction** (Linked Predictive Neural Networks, or LPNN), in which each phoneme class was modeled by a separate neural network, and each network tried to predict the next frame of speech given some recent frames of speech; the prediction errors were used to perform a Viterbi search for the best state sequence, as in an HMM. We found that this approach suffered from a lack of discrimination between the phoneme classes, as all of the networks learned to perform a similar quasi-identity mapping between the quasi-stationary frames of their respective phoneme classes.

The second approach was based on **classification**, in which a single neural network tried to classify a segment of speech into its correct class. This approach proved much more successful, as it naturally supports discrimination between phoneme classes. Within this framework, we explored many variations of the network architecture, input representation, speech model, training procedure, and testing procedure. From these experiments, we reached the following primary conclusions:

- **Outputs as posterior probabilities.** The output activations of a classification network form highly accurate estimates of the posterior probabilities $P(class|input)$, in agreement with theory. Furthermore, these posteriors can be converted into likelihoods $P(input|class)$ for more effective Viterbi search, by simply dividing the activations by the class priors $P(class)$, in accordance with Bayes Rule[1]. Intuitively, we note that the priors should be factored out from the posteriors because they are already reflected in the language model (lexicon plus grammar) used during testing.
- **MLP vs. TDNN.** A simple MLP yields better word accuracy than a TDNN with the same inputs and outputs[2], when each is trained as a frame classifier using a large database. This can be explained in terms of a tradeoff between the degree of hierarchy in a network's time delays, vs. the trainability of the network. As time delays are redistributed higher within a network, each hidden unit sees less context, so it becomes a simpler, less potentially powerful pattern recognizer; however, it also receives more training because it is applied over several adjacent positions (with tied weights), so it learns its simpler patterns more reliably. Thus, when relatively little training data is available — as in early experiments in phoneme recognition (Lang 1989, Waibel et al 1989) — hierarchical time delays serve to increase the amount of training data per weight and improve the system's accuracy. On the other hand, when a large amount of training data is available — as in our CSR experiments — a TDNN's hierarchical time delays make the hidden units unnecessarily coarse and hence degrade the system's accuracy, so a simple MLP becomes preferable.

1. The remaining factor of $P(input)$ can be ignored during recognition, since it is a constant for all classes in a given frame.
2. Here we define a "simple MLP" as an MLP with time delays only in the input layer, and a "TDNN" as an MLP with time delays distributed hierarchically (ignoring the temporal integration layer of the classical TDNN).

- **Word level training.** Word-level training, in which error is backpropagated from a word-level unit that receives its input from the phoneme layer according to a DTW alignment path, yields better results than frame-level or phoneme-level training, because it enhances the consistency between the training criterion and testing criterion. Word-level training increases the system's word accuracy even if the network contains no additional trainable weights; but if the additional weights are trainable, the accuracy improves still further.
- **Adaptive learning rate schedule.** The learning rate schedule is critically important for a neural network. No predetermined learning rate schedule can always give optimal results, so we developed an adaptive technique which searches for the optimal schedule by trying various learning rates and retaining the one that yields the best cross validation results in each iteration of training. This search technique yielded learning rate schedules that generally decreased with each iteration, but which always gave better results than any fixed schedule that tried to approximate the schedule's trajectory.
- **Input representation.** In theory, neural networks do not require careful preprocessing of the input data, since they can automatically learn any useful transformations of the data; but in practice, such preprocessing helps a network to learn somewhat more effectively. For example, delta inputs are theoretically unnecessary if a network is already looking at a window of input frames, but they are helpful anyway because they save the network the trouble of learning to compute the temporal dynamics. Similarly, a network can learn more efficiently if its input space is first orthogonalized by a technique such as Linear Discriminant Analysis. For this reason, in a comparison between various input representations, we obtained best results with a window of spectral and delta-spectral coefficients, orthogonalized by LDA.
- **Gender dependence.** Speaker-independent accuracy can be improved by training separate networks on separate clusters of speakers, and mixing their results during testing according to an automatic identification of the unknown speaker's cluster. This technique is helpful because it separates and hence reduces the overlap in distributions that come from different speaker clusters. We found, in particular, that using two separate gender-dependent networks gives a substantial increase in accuracy, since there is a clear difference between male and female speaker characteristics, and a speaker's gender can be identified by a neural network with near-perfect accuracy.

9.3. Advantages of NN-HMM hybrids

Finally, NN-HMM hybrids offer several theoretical advantages over standard HMM speech recognizers. Specifically:

- **Modeling accuracy.** Discrete density HMMs suffer from quantization errors in their input space, while continuous or semi-continuous density HMMs suffer from *model mismatch*, i.e., a poor match between the a priori choice of statistical model (e.g., a mixture of K Gaussians) and the true density of acoustic space. By contrast, neural networks are nonparametric models that neither suffer from quantization error nor make detailed assumptions about the form of the distribution to be modeled. Thus a neural network can form more accurate acoustic models than an HMM.
- **Context sensitivity.** HMMs assume that speech frames are independent of each other, so they examine only one frame at a time. In order to take advantage of contextual information in neighboring frames, HMMs must artificially absorb those frames into the current frame (e.g., by introducing multiple streams of data in order to exploit delta coefficients, or using LDA to transform these streams into a single stream). By contrast, neural networks can naturally accommodate any size input window, because the number of weights required in a network simply grows linearly with the number of inputs. Thus a neural network is naturally more context sensitive than an HMM.
- **Discrimination**. The standard HMM training criterion, Maximum Likelihood, does not explicitly discriminate between acoustic models, hence the models are not optimized for the essentially discriminative task of word recognition. It is possible to improve discrimination in an HMM by using the Maximum Mutual Information criterion, but this is more complex and difficult to implement properly. By contrast, discrimination is a natural property of neural networks when they are trained to perform classification. Thus a neural network can discriminate more naturally than an HMM.
- **Economy.** An HMM uses its parameters to model the surface of the density function in acoustic space, in terms of the likelihoods $P(input|class)$. By contrast, a neural network uses its parameters to model the boundaries between acoustic classes, in terms of the posteriors $P(class|input)$. Either surfaces or boundaries can be used for classifying speech, but boundaries require fewer parameters and thus can make better use of limited training data. For example, we have achieved 90.5% accuracy using only about 67,000 parameters, while Sphinx obtained only 84.4% accuracy using 111,000 parameters (Lee 1988), and SRI's DECIPHER obtained only 86.0% accuracy using 125,000 parameters (Renals et al 1992). Thus a neural network is more economical than an HMM.

HMMs are also known to be handicapped by their First-Order Assumption, i.e., the assumption that all probabilities depend solely on the current state, independent of previous history; this limits the HMM's ability to model coarticulatory effects, or to model durations accurately. Unfortunately, NN-HMM hybrids share this handicap, because the First-Order Assumption is a property of the HMM temporal model, not of the NN acoustic model. We believe that further research into connectionism could eventually lead to new and powerful techniques for temporal pattern recognition based on neural networks. If and when that happens, it may become possible to design systems that are based entirely on neural networks, potentially further advancing the state of the art in speech recognition.

Appendix A. Final System Design

Our best results with context independent phoneme models — 90.5% word accuracy on the speaker independent Resource Management database — were obtained by a NN-HMM hybrid with the following design:

- Network architecture:
    - Inputs:
        - 16 LDA coefficients per frame, derived from 16 melscale spectral plus 16 delta-spectral coefficients.
        - 9 frame window, with delays = -4...+4
        - Inputs scaled to [-1,+1].
    - Hidden layer:
        - 100 hidden units.
        - Each unit receives input from all input units.
        - Unit activation = tanh (net input) = [-1,+1].
    - Phoneme layer:
        - 61 phoneme units.
        - Each unit receives input from all hidden units.
        - Unit activation = softmax (net input) = [0,1].
    - DTW layer:
        - 6429 units, corresponding to pronunciations of all 994 words.
        - Each unit receives input from one phoneme unit.
        - Unit activation = linear, equal to net input.
    - Word layer:
        - 994 units, one per word.
        - Each unit receives input from DTW units along alignment path.
        - Unit activation = linear, equal to DTW path score / duration.
    - Weights:
        - All weights below the DTW layer are trainable.
        - Initial weights = randomized in range 1.0 / sqrt(fanin).
        - Biases are initialized like the weights.

- Phoneme model:
    - 61 TIMIT phonemes.
    - 1 state per phoneme.

- Training:
  - Database = Resource Management.
  - Training set = 2590 sentences (male), or 1060 sentences (female).
  - Cross validation set = 240 sentences (male), or 100 sentences (female).
  - Labels = generated by Viterbi alignment using a well-trained NN-HMM.
  - Learning rate schedule = based on search and cross validation results.
  - No momentum, no derivative offset.
  - Bootstrap phase:
    - Frame level training (7 iterations).
    - Frames presented in random order, based on random selection with replacement from whole training set.
    - Weights updated after each frame.
    - Phoneme targets = 0.0 or 1.0.
    - Error criterion = Cross Entropy.
  - Final phase:
    - Word level training (2 iterations).
    - Sentences presented in random order.
    - Frames presented in normal order within each sentence.
    - Weights updated after each sentence.
    - Word targets = 0.0 or 1.0.
    - Error criterion = Classification Figure of Merit.
    - Error backpropagated only if within 0.3 of correct output.

- Testing:
  - Test set = 600 sentences = Feb89 & Oct89 test sets.
  - Grammar = word pairs $\Rightarrow$ perplexity 60.
  - One pronunciation per word in the dictionary.
  - Viterbi search using log (Y_i / P_i), where
    Y_i = network output activation of phoneme i,
    P_i = prior of phoneme i.
  - Duration constraints:
    - Minimum:
      - 1/2 average duration per phoneme.
      - implemented via state duplication.
    - Maximum = none.
  - Word transition penalty = -15 (additive penalty).
  - Results: 90.5% word accuracy.

Appendix B. Proof that Classifier Networks Estimate Posterior Probabilities

It was recently discovered that if a multilayer perceptron is asymptotically trained as a 1-of-N classifier using the mean squared error (MSE) criterion, then its output activations will approximate the posterior class probability $P(class|input)$, with an accuracy that improves with the size of the training set. This important fact has been proven by Gish (1990), Bourlard & Wellekens (1990), Hampshire & Pearlmutter (1990), Richard and Lippmann (1991), Ney (1991), and others. The following is a proof due to Ney.

Proof. Assume that a classifier network is trained on a vast population of training samples (x,c) from distribution $p(x,c)$, where x is the input and c is its correct class. (Note that the same input x in different training samples may belong to different classes $\{c\}$, since classes may overlap.) The network computes the function $g_k(x)$ = the activation of the kth output unit. Output targets are $T_{kc} = 1$ when $k = c$ or 0 when $k \neq c$. Training with the squared error criterion minimizes this error in proportion to the density of the training sample space:

$$E = \int_x \sum_c p(x, c) \sum_k (T_{kc} - g_k(x))^2 \tag{76}$$

$$= \int_x \sum_k \sum_c p(x, c) \; (T_{kc} - g_k(x))^2 \tag{77}$$

$$= \int_x \sum_k E_{xk} \tag{78}$$

where

$$E_{xk} = \sum_c p(x, c) \; (T_{kc} - g_k(x))^2 \tag{79}$$

Splitting this into two cases, i.e., $c = k$ and $c \neq k$, we obtain

$$E_{xk} = p(x,k)\,(1-g_k(x))^2 + \sum_{c \neq k} p(x,c)\,(0-g_k(x))^2 \tag{80}$$

$$= p(x,k)\left(1-2g_k(x)+g_k^2(x)\right) + \left(p(x)-p(x,k)\right)\left(g_k^2(x)\right) \tag{81}$$

$$= p(x,k) - 2p(x,k)\,g_k(x) + p(x)\,g_k^2(x) \tag{82}$$

Since $p(x,k) = p(k|x) \cdot p(x)$, an algebraic expansion will show that the above is equivalent to

$$E_{xk} = p(x)\left[p(k|x)-g_k(x)\right]^2 - p(x,k)\left[1-p(k|x)\right] \tag{83}$$

which is minimized when $g_k(x) = P(k|x)$, i.e., when the output activation equals the posterior class probability. ■

Hampshire and Pearlmutter (1990) generalized this proof, showing that the same conclusion holds for a network trained by any of the standard error criteria based on target vectors, e.g., Mean Squared Error, Cross Entropy, McClelland Error, etc.

Bibliography

[1] Ackley, D., Hinton, G., and Sejnowski, T. (1985). A Learning Algorithm for Boltzmann Machines. *Cognitive Science* 9, 147-169. Reprinted in Anderson and Rosenfeld (1988).

[2] Anderson, J. and Rosenfeld, E. (1988). Neurocomputing: Foundations of Research. Cambridge: MIT Press.

[3] Austin, S., Zavaliagkos, G., Makhoul, J., and Schwartz, R. (1992). Speech Recognition Using Segmental Neural Nets. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[4] Bahl, L., Bakis, R., Cohen, P., Cole, A., Jelinek, F., Lewis, B., and Mercer, R. (1981). Speech Recognition of a Natural Text Read as Isolated Words. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1981.*

[5] Bahl, L., Brown, P., De Souza, P., and Mercer, R. (1988). Speech Recognition with Continuous-Parameter Hidden Markov Models. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1988.*

[6] Barnard, E. (1992). Optimization for Training Neural Networks. *IEEE Trans. on Neural Networks,* 3(2), March 1992.

[7] Barto, A., and Anandan, P. (1985). Pattern Recognizing Stochastic Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics* 15, 360-375.

[8] Bellagarda, J. and Nahamoo, D. (1988). Tied-Mixture Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1988.*

[9] Bengio, Y., DeMori, R., Flammia, G., and Kompe, R. (1992). Global Optimization of a Neural Network-Hidden Markov Model Hybrid. *IEEE Trans. on Neural Networks*, 3(2):252-9, March 1992.

[10] Bodenhausen, U., and Manke, S. (1993). Connectionist Architectural Learning for High Performance Character and Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[11] Bodenhausen, U. (1994). Automatic Structuring of Neural Networks for Spatio-Temporal Real-World Applications. PhD Thesis, University of Karlsruhe, Germany.

[12] Bourlard, H. and Wellekens, C. (1990). Links Between Markov Models and Multilayer Perceptrons. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(12), December 1990. Originally appeared as Technical Report Manuscript M-263, Philips Research Laboratory, Brussels, Belgium, 1988.

[13] Bourlard, H. and Morgan, N. (1990). A Continuous Speech Recognition System Embedding MLP into HMM. In *Advances in Neural Information Processing Systems 2*, Touretzky, D. (ed.), Morgan Kaufmann Publishers.

[14] Bourlard, H., Morgan, N., Wooters, C., and Renals, S. (1992). CDNN: A Context Dependent Neural Network for Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[15] Bourlard, H. and Morgan, N. (1994). Connectionist Speech Recognition: A Hybrid Approach. Kluwer Academic Publishers.

[16] Bregler, C., Hild, H., Manke, S., and Waibel, A. (1993). Improving Connected Letter Recognition by Lipreading. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[17] Bridle, J. (1990). Alpha-Nets: A Recurrent "Neural" Network Architecture with a Hidden Markov Model Interpretation. *Speech Communication*, 9:83-92, 1990.

[18] Brown, P. (1987). The Acoustic-Modeling Problem in Automatic Speech Recognition. PhD Thesis, Carnegie Mellon University.

[19] Burr, D. (1988). Experiments on Neural Net Recognition of Spoken and Written Text. In *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 36, 1162-1168.

[20] Burton, D., Shore, J., and Buck, J. (1985). Isolated-Word Speech Recognition Using Multisection Vector Quantization Codebooks. In *IEEE Trans. on Acoustics, Speech and Signal Processing*, 33, 837-849.

[21] Cajal, S. (1892). A New Concept of the Histology of the Central Nervous System. In Rottenberg and Hochberg (eds.), Neurological Classics in Modern Translation. New York: Hafner, 1977.

[22] Carpenter, G. and Grossberg, S. (1988). The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. *Computer* 21(3), March 1988.

[23] Cybenko, G. (1989). Approximation by Superpositions of a Sigmoid Function. *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303-314.

[24] De La Noue, P., Levinson, S., and Sondhi, M. (1989). Incorporating the Time Correlation Between Successive Observations in an Acoustic-Phonetic Hidden Markov Model for Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1987.*

[25] Doddington, G. (1989). Phonetically Sensitive Discriminants for Improved Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1989.*

[26] Duda, R. and Hart, P. (1973). Pattern Classification and Scene Analysis. New York: Wiley.

[27] Elman, J. and Zipser, D. (1987). Learning the Hidden Structure of Speech. ICS Report 8701, Institute for Cognitive Science, University of California, San Diego, La Jolla, CA.

[28] Elman, J. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179-211, 1990.

[29] Fahlman, S. (1988). An Empirical Study of Learning Speed in Back-Propagation Networks. Technical Report CMU-CS-88-162, Carnegie Mellon University.

[30] Fahlman, S. and Lebiere, C. (1990). The Cascade-Correlation Learning Architecture. in *Advances in Neural Information Processing Systems 2*, Touretzky, D. (ed.), Morgan Kaufmann Publishers, Los Altos CA, pp. 524-532.

[31] Fodor, J. and Pylyshyn, Z. (1988). Connectionism and Cognitive Architecture: A Critical Analysis. In Pinker and Mehler (eds.), *Connections and Symbols*, MIT Press, 1988.

[32] Franzini, M., Witbrock, M., and Lee, K.F. (1989). Speaker-Independent Recognition of Connected Utterances using Recurrent and Non-Recurrent Neural Networks. In *Proc. International Joint Conference on Neural Networks*, 1989.

[33] Franzini, M., Lee, K.F., and Waibel, A. (1990). Connectionist Viterbi Training: A New Hybrid Method for Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[34] Furui, S. (1993). Towards Robust Speech Recognition Under Adverse Conditions. In *Proc. of the ESCA Workshop on Speech Processing and Adverse Conditions*, pp. 31-41, Cannes-Mandelieu, France.

[35] Gish, H. (1990). A Probabilistic Approach to the Understanding and Training of Neural Network Classifiers. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[36] Gold, B. (1988). A Neural Network for Isolated Word Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1988.*

[37] Haffner, P., Franzini, M., and Waibel, A. (1991). Integrating Time Alignment and Connectionist Networks for High Performance Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[38] Haffner, P., and Waibel, A. (1992). Multi-State Time Delay Neural Networks for Continuous Speech Recognition. In *Advances in Neural Information Processing Systems 4*, Moody, J., Hanson, S., Lippmann, R. (eds), Morgan Kaufmann Publishers.

[39] Hampshire, J. and Waibel, A. (1990). The Meta-Pi Network: Connectionist Rapid Adaptation for High-Performance Multi-Speaker Phoneme Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[40] Hampshire, J. and Waibel, A. (1990a). A Novel Objective Function for Improved Phoneme Recognition using Time Delay Neural Networks. *IEEE Trans. on Neural Networks*, 1(2), June 1990.

[41] Hampshire, J. and Pearlmutter, B. (1990). Equivalence Proofs for Multi-Layer Perceptron Classifiers and the Bayesian Discriminant Function. In *Proc. of the 1990 Connectionist Models Summer School*, Morgan Kaufmann Publishers.

[42] Hassibi, B., and Stork, D. (1993). Second Order Derivative for Network Pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems 5*, Hanson, S., Cowan, J., and Giles, C.L. (eds), Morgan Kaufmann Publishers.

[43] Hebb, D. (1949). The Organization of Behavior. New York: Wiley. Partially reprinted in Anderson and Rosenfeld (1988).

[44] Hermansky, H. (1990). Perceptual Linear Predictive (PLP) Analysis of Speech. *Journal of the Acoustical Society of America*, 87(4):1738-52, 1990.

[45] Hertz, J., Krogh, A., and Palmer, R. (1991). Introduction to the Theory of Neural Computation. Addison-Wesley.

[46] Hild, H. and Waibel, A. (1993). Connected Letter Recognition with a Multi-State Time Delay Neural Network. In *Advances in Neural Information Processing Systems 5*, Hanson, S., Cowan, J., and Giles, C.L. (eds), Margan Kaufmann Publishers.

[47] Hinton, G. (1989). Connectionist Learning Procedures. *Artificial Intelligence* 40:1(3), 185-235.

[48] Hofstadter, D. (1979). Godel, Escher, Bach: An Eternal Golden Braid. Basic Books.

[49] Hopfield, J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. National Academy of Sciences USA,* 79:2554-58, April 1982. Reprinted in Anderson and Rosenfeld (1988).

[50] Huang, W.M. and Lippmann, R. (1988). Neural Net and Traditional Classifiers. In *Neural Information Processing Systems*, Anderson, D. (ed.), 387-396. New York: American Institute of Physics.

[51] Huang, X.D. (1992). Phoneme Classification using Semicontinuous Hidden Markov Models. *IEEE Trans. on Signal Processing,* 40(5), May 1992.

[52] Huang, X.D. (1992a). Speaker Normalization for Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[53] Hwang, M.Y. and Huang, X.D. (1993). Shared-Distribution Hidden Markov Models for Speech Recognition. *IEEE Trans. on Speech and Audio Processing*, vol.1, 1993, pp 414-420.

[54] Hwang, M.Y., Huang, X.D., and Alleva, F. (1993b). Predicting Unseen Triphones with Senones. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[55] Idan, Y., Auger, J., Darbel, N., Sales, M., Chevallier, R., Dorizzi, B., and Cazuguel, G. (1992). Comparative Study of Neural Networks and Non-Parametric Statistical Methods for Off-Line Handwritten Character Recognition. In *Proc. International Conference on Artificial Neural Networks*, 1992.

[56] Iso, K. and Watanabe, T. (1990). Speaker-Independent Word Recognition using a Neural Prediction Model. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[57] Iso, K. and Watanabe, T. (1991). Large Vocabulary Speech Recognition using Neural Prediction Model. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[58] Itakura, F. (1975). Minimum Prediction Residual Principle Applied to Speech Recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 23(1):67-72, February 1975. Reprinted in Waibel and Lee (1990).

[59] Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive Mixtures of Local Experts. *Neural Computation* 3(1), 79-87.

[60] Jain, A., Waibel, A., and Touretzky, D. (1992). PARSEC: A Structured Connectionist Parsing System for Spoken Language. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[61] Jolliffe, I. (1986). Principle Component Analysis. New York: Springer-Verlag.

[62] Jordan, M. (1986). Serial Order: A Parallel Distributed Processing Approach. ICS Technical Report 8604, UCSD.

[63] Kammerer, B. and Kupper, W. (1988). Experiments for Isolated-Word Recognition with Single and Multi-Layer Perceptrons. Abstracts of 1st Annual INNS Meeting, Boston.

[64] Kimura, S. (1990). 100,000-Word Recognition Using Acoustic-Segment Networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[65] Kohonen, T. (1989). Self-Organization and Associative Memory (3rd edition). Berlin: Springer-Verlag.

[66] Konig, Y. and Morgan, N. (1993). Supervised and Unsupervised Clustering of the Speaker Space for Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[67] Krishnaiah, P. and Kanal, L., eds. (1982). Classification, Pattern Recognition, and Reduction of Dimensionality. Handbook of Statistics, vol. 2. Amsterdam: North Holland.

[68] Krogh, A. and Hertz, J. (1992). A Simple Weight Decay Can Improve Generalization. In *Advances In Neural Information Processing Systems 4*, Moody, J., Hanson, S., Lippmann, R. (eds), Morgan Kaufmann Publishers.

[69] Kubala, F. and Schwartz, R. (1991). A New Paradigm for Speaker-Independent Training. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[70] Lang, K. (1989). A Time-Delay Neural Network Architecture for Speech Recognition. PhD Thesis, Carnegie Mellon University.

[71] Lang, K., Waibel, A., and Hinton, G. (1990). A Time-Delay Neural Network Architecture for Isolated Word Recognition. *Neural Networks* 3(1): 23-43.

[72] Le Cun, Y., Matan, O., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jacket, L., and Baird, H. (1990). Handwritten ZIP Code Recognition with Multilayer Networks. In *Proc. 10th International Conference on Pattern Recognition*, June 1990.

[73] LeCun, Y., Denker, J., and Solla, S. (1990b). Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*, Touretzky, D. (ed), Morgan Kaufmann Publishers.

[74] Lee, K.F. (1988). Large Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System. PhD Thesis, Carnegie Mellon University.

[75] Levin, E. (1990). Word Recognition using Hidden Control Neural Architecture. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[76] Linsker, R. (1986). From Basic Network Principles to Neural Architecture. *Proc. National Academy of Sciences, USA* 83, 7508-12, 8390-94, 8779-83.

[77] Lippmann, R. and Gold, B. (1987). Neural Classifiers Useful for Speech Recognition. In *1st International Conference on Neural Networks, IEEE.*

[78] Lippmann, R. (1989). Review of Neural Networks for Speech Recognition. *Neural Computation* 1(1):1-38, Spring 1989. Reprinted in Waibel and Lee (1990).

[79] Lippmann, R. and Singer, E. (1993). Hybrid Neural Network/HMM Approaches to Wordspotting. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[80] McCulloch, W. and Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* 5: 115-133. Reprinted in Anderson and Rosenfeld (1988).

[81] McDermott, E. and Katagiri, S. (1991). LVQ-Based Shift-Tolerant Phoneme Recognition. *IEEE Trans. on Signal Processing*, 39(6):1398-1411, June 1991.

[82] Mellouk, A. and Gallinari, P. (1993). A Discriminative Neural Prediction System for Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[83] Minsky, M. (1967). Computation: Finite and Infinite Machines. Englewood Cliffs: Prentice-Hall.

[84] Minsky, M. and Papert, S. (1969). Perceptrons. Cambridge: MIT Press. Partially reprinted in Anderson and Rosenfeld (1988).

[85] Miyatake, M., Sawai, H., and Shikano, K. (1990). Integrated Training for Spotting Japanese Phonemes Using Large Phonemic Time-Delay Neural Networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[86] Moody, J. and Darken, C. (1989). Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation* 1(2), 281-294.

[87] Morgan, D., Scofield, C., and Adcock, J. (1991). Multiple Neural Network Topologies Applied to Keyword Spotting. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[88] Morgan, N. and Bourlard, H. (1990). Continuous Speech Recognition using Multilayer Perceptrons with Hidden Markov Models. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[89] Munro, P. (1987). A Dual Back-Propagation Scheme for Scalar Reward Learning. In *The Ninth Annual Conference of the Cognitive Science Society* (Seattle 1987), 165-176. Hillsdale: Erlbaum.

[90] Ney, H. (1984). The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 32(2):263-271, April 1984. Reprinted in Waibel and Lee (1990).

[91] Ney, H. and Noll, A. (1988). Phoneme Modeling using Continuous Mixture Densities. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1988.*

[92] Ney, H. (1991). Speech Recognition in a Neural Network Framework: Discriminative Training of Gaussian Models and Mixture Densities as Radial Basis Functions. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[93] Osterholtz, L., Augustine, C., McNair, A., Rogina, I., Saito, H., Sloboda, T., Tebelskis, J., and Waibel, A. (1992). Testing Generality in Janus: A Multi-Lingual Speech Translation System. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[94] Peeling, S. and Moore, R. (1987). Experiments in Isolated Digit Recognition Using the Multi-Layer Perceptron. Technical Report 4073, Royal Speech and Radar Establishment, Malvern, Worcester, Great Britain.

[95] Petek, B., Waibel, A., and Tebelskis, J. (1991). Integrated Phoneme-Function Word Architecture of Hidden Control Neural Networks for Continuous Speech Recognition. In *Proc. European Conference on Speech Communication and Technology*, 1991.

[96] Petek, B. and Tebelskis, J. (1992). Context-Dependent Hidden Control Neural Network Architecture for Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[97] Pinker, S. and Prince, A. (1988). On Language and Connectionism. In Pinker and Mehler (eds.), *Connections and Symbols*, MIT Press, 1988.

[98] Pomerleau, D. (1993). Neural Network Perception for Mobile Robot Guidance. Kluwer Academic Publishing.

[99] Prager, R., Harrison, T., and Fallside, F. (1986). Boltzmann Machines for Speech Recognition. *Computer Speech and Language* 1, 2-27.

[100] Price, P., Fisher, W., Bernstein, J., and Pallett, D. (1988). The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1988.*

[101] Rabiner, L. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2), February 1989. Reprinted in Waibel and Lee (1990).

[102] Rabiner, L. and Juang, B.H. (1993). Fundamentals of Speech Recognition. Prentice Hall.

[103] Reddy, R. (1976). Speech Recognition by Machine: A Review. *Proceedings of the IEEE*, 64(4):502-531, April 1976. Reprinted in Waibel and Lee (1990).

[104] Renals, S., Morgan, N., Cohen, M., and Franco, H. (1992). Connectionist Probability Estimation in the DECIPHER Speech Recognition System. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[105] Richard, M. and Lippmann, R. (1991). Neural Network Classifiers Estimate Bayesian A Posteriori Probabilities. *Neural Computation* 3(4):461-483, Winter 1991.

[106] Robinson, A. and Fallside, F. (1988). Static and Dynamic Error Propagation Networks with Application to Speech Coding. In *Neural Information Processing Systems*, Anderson, D. (ed.), 632-641. New York: American Institute of Physics.

[107] Rosenblatt, F. (1962). Principles of Neurodynamics. New York: Spartan.

[108] Rumelhart, D., McClelland, J., and the PDP Research Group. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press.

[109] Sagisaka, Y., Takeda, K., Katagiri, S., and Kuwabara, H. (1987). Japanese Speech Database with Fine Acoustic-Phonetic Distinctions. Technical Report, ATR Interpreting Telephony Research Laboratory.

[110] Sakoe, H. and Chiba, S. (1978). Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 26(1):43-49, February 1978. Reprinted in Waibel and Lee (1990).

[111] Sakoe, H., Isotani, R., Yoshida, K., and Iso, K. (1989). Speaker-Independent Word Recognition using Dynamic Programming Neural Networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1989.*

[112] Sanger, T. (1989). Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network. *Neural Networks* 2(6), 459-473.

[113] Schmidbauer, O. and Tebelskis, J. (1992). An LVQ Based Reference Model for Speaker Adaptive Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[114] Schwartz, R. and Chow, Y. (1990). The N-Best Algorithm: An Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypothesis. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[115] Sejnowski, T. and Rosenberg, C. (1987). Parallel Networks that Learn to Pronounce English Text. *Complex Systems* 1, 145-168.

[116] Servan-Schreiber, D., Cleeremans, A., and McClelland, J. (1991). Graded State Machines: The Representation of Temporal Contingencies in Simple Recurrent Networks. *Machine Learning*, 7:2-3, 161-193.

[117] Smolensky, P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence* 46(1-2), 159-216.

[118] Sutton, R. (1984). Temporal Credit Assignment in Reinforcement Learning. PhD Thesis, University of Massachusetts, Amherst.

[119] Tank, D. and Hopfield, J. (1987). Neural Computation by Concentrating Information in Time. *Proc. National Academy of Sciences USA*, 84, pp. 1896-1900, April 1987.

[120] Tebelskis, J. and Waibel, A. (1990). Large Vocabulary Recognition using Linked Predictive Neural Networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[121] Tebelskis, J., Waibel, A., Petek, B., and Schmidbauer, O. (1991). Continuous Speech Recognition using Linked Predictive Neural Networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[122] Tebelskis, J. (1993). Performance Through Consistency: Connectionist Large Vocabulary Continuous Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*

[123] Tesauro, G. (1989). Neurogammon Wins Computer Olympiad. *Neural Computation,* 1(3), 321-323.

[124] Tishby, N. (1990). A Dynamical Systems Approach to Speech Processing. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1990.*

[125] Touretzky, D. and Hinton, G. (1988). A Distributed Connectionist Production System. *Cognitive Science* 12(3): 423-466.

[126] Unnikrishnan, K., Hopfield, J., and Tank, D. (1988). Learning Time-Delayed Connections in a Speech Recognition Circuit. *Neural Networks for Computing Conference*, Snowbird, Utah.

[127] Vintsyuk, T. (1971). Element-Wise Recognition of Continuous Speech Composed of Words from a Specified Dictionary. *Kibernetika* 7:133-143, March-April 1971.

[128] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1989). Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 37(3), March 1989. Originally appeared as Technical Report TR-1-0006, ATR Interpreting Telephony Research Laboratories, Japan, 1987. Reprinted in Waibel and Lee (1990).

[129] Waibel, A., Sawai, H., and Shikano, K. (1989a). Modularity and Scaling in Large Phonemic Neural Networks. *IEEE Trans. Acoustics, Speech and Signal Processing*, 37(12): 1888-98.

[130] Waibel, A. and Lee, K.F., eds. (1990). Readings in Speech Recognition. Morgan Kaufmann Publishers.

[131] Waibel, A., Jain, A., McNair, A., Saito, H., Hauptmann, A., and Tebelskis, J. (1991). Janus: A Speech-to-Speech Translation System using Connectionist and Symbolic Processing Strategies. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1991.*

[132] Watrous, R. (1988). Speech Recognition using Connectionist Networks. PhD Thesis, University of Pennsylvania.

[133] Wellekens, C. (1987). Explicit Time Correlation in Hidden Markov Models for Speech Recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1987.*

[134] Witbrock, M. and Haffner, P. (1992). Rapid Connectionist Speaker Adaptation. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[135] Wood, C. (1992). Conference Registration Task for Neural Net Speech Recognition: Speech Collection and Labeling for a Speaker Independent System. Technical Reports CMU-CS-92-197 and CMU-CS-92-198, Carnegie Mellon University.

[136] Woodland, P., Odell, J., Valtchev, V., and Young, S. (1994). Large Vocabulary Continuous Speech Recognition using HTK. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1994.*

[137] Woszczyna, M., Aoki-Waibel, N., Bu, F., Coccaro, N., Horiguchi, K., Kemp, T., Lavie, A., McNair, A., Polzin, T., Rogina, I., Rose, C., Schultz, T., Suhm, B., Tomita, M., and Waibel, A. (1994). Janus 93: Towards Spontaneous Speech Translation. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1994.*

[138] Zeppenfeld, T. and Waibel, A. (1992). A Hybrid Neural Network, Dynamic Programming Word Spotter. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.*

[139] Zeppenfeld, T. and Waibel, A. (1993). Improving the MS-TDNN for Word Spotting. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993.*Cohen, M., Franco, H., Morgan, N., Rumelhart, D., and Abrash, V. (1993). Context-Dependent Multiple Distribution Phonetic Modeling with MLPs. In *Advances in Neural Information Processing Systems 5*, Hanson, S., Cowan, J., and Giles, C.L. (eds), Morgan Kaufmann Publishers.

Author Index

Subject Index