Speech Recognition using Neural Networks

Joe Tebelskis

May 1995
CMU-CS-95-142

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Submitted in partial fulfillment of the requirements for
a degree of Doctor of Philosophy in Computer Science

Thesis Committee:

Alex Waibel, chair
Raj Reddy
Jaime Carbonell
Richard Lippmann, MIT Lincoln Labs

This research was supported during separate phases by ATR Interpreting Telephony Research Laboratories, NEC Corporation, Siemens AG, the National Science Foundation, the Advanced Research Projects Administration, and the Department of Defense under Contract No. MDA904-92-C-5161.

Keywords: Speech recognition, neural networks, hidden Markov models, hybrid systems, acoustic modeling, prediction, classification, probability estimation, discrimination, global optimization.

Abstract

This thesis examines how artificial neural networks can benefit a large vocabulary, speaker independent, continuous speech recognition system. Currently, most speech recognition systems are based on hidden Markov models (HMMs), a statistical framework that supports both acoustic and temporal modeling. Despite their state-of-the-art performance, HMMs make a number of suboptimal modeling assumptions that limit their potential effectiveness. Neural networks avoid many of these assumptions, while they can also learn complex functions, generalize effectively, tolerate noise, and support parallelism. While neural networks can readily be applied to acoustic modeling, it is not yet clear how they can be used for temporal modeling. Therefore, we explore a class of systems called *NN-HMM hybrids*, in which neural networks perform acoustic modeling, and HMMs perform temporal modeling. We argue that a NN-HMM hybrid has several theoretical advantages over a pure HMM system, including better acoustic modeling accuracy, better context sensitivity, more natural discrimination, and a more economical use of parameters. These advantages are confirmed experimentally by a NN-HMM hybrid that we developed, based on context-independent phoneme models, that achieved 90.5% word accuracy on the Resource Management database, in contrast to only 86.0% accuracy achieved by a pure HMM under similar conditions.

In the course of developing this system, we explored two different ways to use neural networks for acoustic modeling: prediction and classification. We found that predictive networks yield poor results because of a lack of discrimination, but classification networks gave excellent results. We verified that, in accordance with theory, the output activations of a classification network form highly accurate estimates of the posterior probabilities $P(class|input)$, and we showed how these can easily be converted to likelihoods $P(input|class)$ for standard HMM recognition algorithms. Finally, this thesis reports how we optimized the accuracy of our system with many natural techniques, such as expanding the input window size, normalizing the inputs, increasing the number of hidden units, converting the network's output activations to log likelihoods, optimizing the learning rate schedule by automatic search, backpropagating error from word level outputs, and using gender dependent networks.

Acknowledgements

I wish to thank Alex Waibel for the guidance, encouragement, and friendship that he managed to extend to me during our six years of collaboration over all those inconvenient oceans — and for his unflagging efforts to provide a world-class, international research environment, which made this thesis possible. Alex's scientific integrity, humane idealism, good cheer, and great ambition have earned him my respect, plus a standing invitation to dinner whenever he next passes through my corner of the world. I also wish to thank Raj Reddy, Jaime Carbonell, and Rich Lippmann for serving on my thesis committee and offering their valuable suggestions, both on my thesis proposal and on this final dissertation. I would also like to thank Scott Fahlman, my first advisor, for channeling my early enthusiasm for neural networks, and teaching me what it means to do good research.

Many colleagues around the world have influenced this thesis, including past and present members of the Boltzmann Group, the NNSpeech Group at CMU, and the NNSpeech Group at the University of Karlsruhe in Germany. I especially want to thank my closest collaborators over these years — Bojan Petek, Otto Schmidbauer, Torsten Zeppenfeld, Hermann Hild, Patrick Haffner, Arthur McNair, Tilo Sloboda, Monika Woszczyna, Ivica Rogina, Michael Finke, and Thorsten Schueler — for their contributions and their friendship. I also wish to acknowledge valuable interactions I've had with many other talented researchers, including Fil Alleva, Uli Bodenhausen, Herve Bourlard, Lin Chase, Mike Cohen, Mark Derthick, Mike Franzini, Paul Gleichauff, John Hampshire, Nobuo Hataoka, Geoff Hinton, Xuedong Huang, Mei-Yuh Hwang, Ken-ichi Iso, Ajay Jain, Yochai Konig, George Lakoff, Kevin Lang, Chris Lebiere, Kai-Fu Lee, Ester Levin, Stefan Manke, Jay McClelland, Chris McConnell, Abdelhamid Mellouk, Nelson Morgan, Barak Pearlmutter, Dave Plaut, Dean Pomerleau, Steve Renals, Roni Rosenfeld, Dave Rumelhart, Dave Sanner, Hidefumi Sawai, David Servan-Schreiber, Bernhard Suhm, Sebastian Thrun, Dave Touretzky, Minh Tue Voh, Wayne Ward, Christoph Windheuser, and Michael Witbrock. I am especially indebted to Yochai Konig at ICSI, who was extremely generous in helping me to understand and reproduce ICSI's experimental results; and to Arthur McNair for taking over the Janus demos in 1992 so that I could focus on my speech research, and for constantly keeping our environment running so smoothly. Thanks to Hal McCarter and his colleagues at Adaptive Solutions for their assistance with the CNAPS parallel computer; and to Nigel Goddard at the Pittsburgh Supercomputer Center for help with the Cray C90. Thanks to Roni Rosenfeld, Lin Chase, and Michael Finke for proofreading portions of this thesis.

I am also grateful to Robert Wilensky for getting me started in Artificial Intelligence, and especially to both Douglas Hofstadter and Allen Newell for sharing some treasured, pivotal hours with me.

Many friends helped me maintain my sanity during the PhD program, as I felt myself drowning in this overambitious thesis. I wish to express my love and gratitude especially to Bart Reynolds, Sara Fried, Mellen Lovrin, Pam Westin, Marilyn & Pete Fast, Susan Wheeler, Gowthami Rajendran, I-Chen Wu, Roni Rosenfeld, Simona & George Necula, Francesmary Modugno, Jade Goldstein, Hermann Hild, Michael Finke, Kathie Porsche, Phyllis Reuther, Barbara White, Bojan & Davorina Petek, Anne & Scott Westbrook, Richard Weinapple, Marv Parsons, and Jeanne Sheldon. I have also prized the friendship of Catherine Copetas, Prasad Tadepalli, Hanna Djajapranata, Arthur McNair, Torsten Zeppenfeld, Tilo Sloboda, Patrick Haffner, Mark Maimone, Spiro Michaylov, Prasad Chalisani, Angela Hickman, Lin Chase, Steve Lawson, Dennis & Bonnie Lunder, and too many others to list. Without the support of my friends, I might not have finished the PhD.

I wish to thank my parents, Virginia and Robert Tebelskis, for having raised me in such a stable and loving environment, which has enabled me to come so far. I also thank the rest of my family & relatives for their love.

This thesis is dedicated to Douglas Hofstadter, whose book “Godel, Escher, Bach” changed my life by suggesting how consciousness can emerge from subsymbolic computation, shaping my deepest beliefs and inspiring me to study Connectionism; and to the late Allen Newell, whose genius, passion, warmth, and humanity made him a beloved role model whom I could only dream of emulating, and whom I now sorely miss.

Table of Contents

1. Introduction

Speech is a natural mode of communication for people. We learn all the relevant skills during early childhood, without instruction, and we continue to rely on speech communication throughout our lives. It comes so naturally to us that we don't realize how complex a phenomenon speech is. The human vocal tract and articulators are biological organs with nonlinear properties, whose operation is not just under conscious control but also affected by factors ranging from gender to upbringing to emotional state. As a result, vocalizations can vary widely in terms of their accent, pronunciation, articulation, roughness, nasality, pitch, volume, and speed; moreover, during transmission, our irregular speech patterns can be further distorted by background noise and echoes, as well as electrical characteristics (if telephones or other electronic equipment are used). All these sources of variability make speech recognition, even more than speech generation, a very complex problem.

Yet people are so comfortable with speech that we would also like to interact with our computers via speech, rather than having to resort to primitive interfaces such as keyboards and pointing devices. A speech interface would support many valuable applications — for example, telephone directory assistance, spoken database querying for novice users, "hands-busy" applications in medicine or fieldwork, office dictation devices, or even automatic voice translation into foreign languages. Such tantalizing applications have motivated research in automatic speech recognition since the 1950's. Great progress has been made so far, especially since the 1970's, using a series of engineered approaches that include template matching, knowledge engineering, and statistical modeling. Yet computers are still nowhere near the level of human performance at speech recognition, and it appears that further significant advances will require some new insights.

What makes people so good at recognizing speech? Intriguingly, the human brain is known to be wired differently than a conventional computer; in fact it operates under a radically different computational paradigm. While conventional computers use a very fast & complex central processor with explicit program instructions and locally addressable memory, by contrast the human brain uses a massively parallel collection of slow & simple processing elements (neurons), densely connected by weights (synapses) whose strengths are modified with experience, directly supporting the integration of multiple constraints, and providing a distributed form of associative memory.

The brain's impressive superiority at a wide range of cognitive skills, including speech recognition, has motivated research into its novel computational paradigm since the 1940's, on the assumption that brainlike models may ultimately lead to brainlike performance on many complex tasks. This fascinating research area is now known as *connectionism*, or the study of *artificial neural networks*. The history of this field has been erratic (and laced with

hyperbole), but by the mid-1980's, the field had matured to a point where it became realistic to begin applying connectionist models to difficult tasks like speech recognition. By 1990 (when this thesis was proposed), many researchers had demonstrated the value of neural networks for important subtasks like phoneme recognition and spoken digit recognition, but it was still unclear whether connectionist techniques would scale up to large speech recognition tasks.

This thesis demonstrates that neural networks can indeed form the basis for a general purpose speech recognition system, and that neural networks offer some clear advantages over conventional techniques.

1.1. Speech Recognition

What is the current state of the art in speech recognition? This is a complex question, because a system's accuracy depends on the conditions under which it is evaluated: under sufficiently narrow conditions almost any system can attain human-like accuracy, but it's much harder to achieve good accuracy under general conditions. The conditions of evaluation — and hence the accuracy of any system — can vary along the following dimensions:

- **Vocabulary size and confusability**. As a general rule, it is easy to discriminate among a small set of words, but error rates naturally increase as the vocabulary size grows. For example, the 10 digits "zero" to "nine" can be recognized essentially perfectly (Doddington 1989), but vocabulary sizes of 200, 5000, or 100000 may have error rates of 3%, 7%, or 45% (Itakura 1975, Miyatake 1990, Kimura 1990). On the other hand, even a small vocabulary can be hard to recognize if it contains confusable words. For example, the 26 letters of the English alphabet (treated as 26 "words") are very difficult to discriminate because they contain so many confusable words (most notoriously, the E-set: "B, C, D, E, G, P, T, V, Z"); an 8% error rate is considered good for this vocabulary (Hild & Waibel 1993).
- **Speaker dependence vs. independence.** By definition, a *speaker dependent* system is intended for use by a single speaker, but a *speaker independent* system is intended for use by any speaker. Speaker independence is difficult to achieve because a system's parameters become tuned to the speaker(s) that it was trained on, and these parameters tend to be highly speaker-specific. Error rates are typically 3 to 5 times higher for speaker independent systems than for speaker dependent ones (Lee 1988). Intermediate between speaker dependent and independent systems, there are also *multi-speaker* systems intended for use by a small group of people, and *speaker-adaptive* systems which tune themselves to any speaker given a small amount of their speech as enrollment data.
- **Isolated, discontinuous, or continuous speech.** *Isolated speech* means single words; *discontinuous speech* means full sentences in which words are artificially separated by silence; and *continuous speech* means naturally spoken sentences. Isolated and discontinuous speech recognition is relatively easy because word boundaries are detectable and the words tend to be cleanly pronounced. Continu-

ous speech is more difficult, however, because word boundaries are unclear and their pronunciations are more corrupted by *coarticulation*, or the slurring of speech sounds, which for example causes a phrase like "could you" to sound like "could jou". In a typical evaluation, the word error rates for isolated and continuous speech were 3% and 9%, respectively (Bahl et al 1981).

- **Task and language constraints.** Even with a fixed vocabulary, performance will vary with the nature of constraints on the word sequences that are allowed during recognition. Some constraints may be *task-dependent* (for example, an airline-querying application may dismiss the hypothesis "The apple is red"); other constraints may be *semantic* (rejecting "The apple is angry"), or *syntactic* (rejecting "Red is apple the"). Constraints are often represented by a *grammar*, which ideally filters out unreasonable sentences so that the speech recognizer evaluates only plausible sentences. Grammars are usually rated by their *perplexity*, a number that indicates the grammar's average branching factor (i.e., the number of words that can follow any given word). The difficulty of a task is more reliably measured by its perplexity than by its vocabulary size.
- **Read vs. spontaneous speech.** Systems can be evaluated on speech that is either read from prepared scripts, or speech that is uttered spontaneously. Spontaneous speech is vastly more difficult, because it tends to be peppered with disfluencies like "uh" and "um", false starts, incomplete sentences, stuttering, coughing, and laughter; and moreover, the vocabulary is essentially unlimited, so the system must be able to deal intelligently with unknown words (e.g., detecting and flagging their presence, and adding them to the vocabulary, which may require some interaction with the user).
- **Adverse conditions.** A system's performance can also be degraded by a range of adverse conditions (Furui 1993). These include environmental noise (e.g., noise in a car or a factory); acoustical distortions (e.g, echoes, room acoustics); different microphones (e.g., close-speaking, omnidirectional, or telephone); limited frequency bandwidth (in telephone transmission); and altered speaking manner (shouting, whining, speaking quickly, etc.).

In order to evaluate and compare different systems under well-defined conditions, a number of standardized databases have been created with particular characteristics. For example, one database that has been widely used is the DARPA Resource Management database — a large vocabulary (1000 words), speaker-independent, continuous speech database, consisting of 4000 training sentences in the domain of naval resource management, read from a script and recorded under benign environmental conditions; testing is usually performed using a grammar with a perplexity of 60. Under these controlled conditions, state-of-the-art performance is about 97% word recognition accuracy (or less for simpler systems). We used this database, as well as two smaller ones, in our own research (see Chapter 5).

The central issue in speech recognition is dealing with variability. Currently, speech recognition systems distinguish between two kinds of variability: acoustic and temporal. *Acoustic variability* covers different accents, pronunciations, pitches, volumes, and so on,

while *temporal variability* covers different speaking rates. These two dimensions are not completely independent — when a person speaks quickly, his acoustical patterns become distorted as well — but it's a useful simplification to treat them independently.

Of these two dimensions, temporal variability is easier to handle. An early approach to temporal variability was to linearly stretch or shrink (*"warp"*) an unknown utterance to the duration of a known template. Linear warping proved inadequate, however, because utterances can accelerate or decelerate at any time; instead, nonlinear warping was obviously required. Soon an efficient algorithm known as *Dynamic Time Warping* was proposed as a solution to this problem. This algorithm (in some form) is now used in virtually every speech recognition system, and the problem of temporal variability is considered to be largely solved[1].

Acoustic variability is more difficult to model, partly because it is so heterogeneous in nature. Consequently, research in speech recognition has largely focused on efforts to model acoustic variability. Past approaches to speech recognition have fallen into three main categories:

1. **Template-based approaches**, in which unknown speech is compared against a set of prerecorded words (*templates*), in order to find the best match. This has the advantage of using perfectly accurate word models; but it also has the disadvantage that the prerecorded templates are fixed, so variations in speech can only be modeled by using many templates per word, which eventually becomes impractical.

2. **Knowledge-based approaches**, in which "expert" knowledge about variations in speech is hand-coded into a system. This has the advantage of explicitly modeling variations in speech; but unfortunately such expert knowledge is difficult to obtain and use successfully, so this approach was judged to be impractical, and automatic learning procedures were sought instead.

3. **Statistical-based approaches**, in which variations in speech are modeled statistically (e.g., by *Hidden Markov Models*, or *HMMs*), using automatic learning procedures. This approach represents the current state of the art. The main disadvantage of statistical models is that they must make a priori modeling assumptions, which are liable to be inaccurate, handicapping the system's performance. We will see that neural networks help to avoid this problem.

1.2. Neural Networks

Connectionism, or the study of artificial neural networks, was initially inspired by neurobiology, but it has since become a very interdisciplinary field, spanning computer science, electrical engineering, mathematics, physics, psychology, and linguistics as well. Some researchers are still studying the neurophysiology of the human brain, but much attention is

1. Although there remain unresolved secondary issues of duration constraints, speaker-dependent speaking rates, etc.

now being focused on the general properties of neural computation, using simplified neural models. These properties include:

- **Trainability.** Networks can be taught to form associations between any input and output patterns. This can be used, for example, to teach the network to classify speech patterns into phoneme categories.
- **Generalization.** Networks don't just memorize the training data; rather, they learn the underlying patterns, so they can generalize from the training data to new examples. This is essential in speech recognition, because acoustical patterns are never exactly the same.
- **Nonlinearity.** Networks can compute nonlinear, nonparametric functions of their input, enabling them to perform arbitrarily complex transformations of data. This is useful since speech is a highly nonlinear process.
- **Robustness.** Networks are tolerant of both physical damage and noisy data; in fact noisy data can help the networks to form better generalizations. This is a valuable feature, because speech patterns are notoriously noisy.
- **Uniformity.** Networks offer a uniform computational paradigm which can easily integrate constraints from different types of inputs. This makes it easy to use both basic and differential speech inputs, for example, or to combine acoustic and visual cues in a multimodal system.
- **Parallelism.** Networks are highly parallel in nature, so they are well-suited to implementations on massively parallel computers. This will ultimately permit very fast processing of speech or other data.

There are many types of connectionist models, with different architectures, training procedures, and applications, but they are all based on some common principles. An artificial neural network consists of a potentially large number of simple processing elements (called *units*, *nodes*, or *neurons*), which influence each other's behavior via a network of excitatory or inhibitory weights. Each unit simply computes a nonlinear weighted sum of its inputs, and broadcasts the result over its outgoing connections to other units. A training set consists of patterns of values that are assigned to designated input and/or output units. As patterns are presented from the training set, a learning rule modifies the strengths of the weights so that the network gradually learns the training set. This basic paradigm[1] can be fleshed out in many different ways, so that different types of networks can learn to compute implicit functions from input to output vectors, or automatically cluster input data, or generate compact representations of data, or provide content-addressable memory and perform pattern completion.

1. Many biological details are ignored in these simplified models. For example, biological neurons produce a sequence of pulses rather than a stable activation value; there exist several different types of biological neurons; their physical geometry can affect their computational behavior; they operate asynchronously, and have different cycle times; and their behavior is affected by hormones and other chemicals. Such details may ultimately prove necessary for modeling the brain's behavior, but for now even the simplified model has enough computational power to support very interesting research.

Neural networks are usually used to perform static pattern recognition, that is, to statically map complex inputs to simple outputs, such as an N-ary classification of the input patterns. Moreover, the most common way to train a neural network for this task is via a procedure called *backpropagation* (Rumelhart et al, 1986), whereby the network's weights are modified in proportion to their contribution to the observed error in the output unit activations (relative to desired outputs). To date, there have been many successful applications of neural networks trained by backpropagation. For instance:

- *NETtalk* (Sejnowski and Rosenberg, 1987) is a neural network that learns how to pronounce English text. Its input is a window of 7 characters (orthographic text symbols), scanning a larger text buffer, and its output is a phoneme code (relayed to a speech synthesizer) that tells how to pronounce the middle character in that context. During successive cycles of training on 1024 words and their pronunciations, NETtalk steadily improved is performance like a child learning how to talk, and it eventually produced quite intelligible speech, even on words that it had never seen before.
- *Neurogammon* (Tesauro 1989) is a neural network that learns a winning strategy for Backgammon. Its input describes the current position, the dice values, and a possible move, and its output represents the merit of that move, according to a training set of 3000 examples hand-scored by an expert player. After sufficient training, the network generalized well enough to win the gold medal at the computer olympiad in London, 1989, defeating five commercial and two non-commercial programs, although it lost to a human expert.
- *ALVINN* (Pomerleau 1993) is a neural network that learns how to drive a car. Its input is a coarse visual image of the road ahead (provided by a video camera and an imaging laser rangefinder), and its output is a continuous vector that indicates which way to turn the steering wheel. The system learns how to drive by observing how a person drives. ALVINN has successfully driven at speeds of up to 70 miles per hour for more than 90 miles, under a variety of different road conditions.
- *Handwriting recognition* (Le Cun et al, 1990) based on neural networks has been used to read ZIP codes on US mail envelopes. Size-normalized images of isolated digits, found by conventional algorithms, are fed to a highly constrained neural network, which transforms each visual image to one of 10 class outputs. This system has achieved 92% digit recognition accuracy on actual mail provided by the US Postal Service. A more elaborate system by Bodenhausen and Manke (1993) has achieved up to 99.5% digit recognition accuracy on another database.

Speech recognition, of course, has been another proving ground for neural networks. Researchers quickly achieved excellent results in such basic tasks as voiced/unvoiced discrimination (Watrous 1988), phoneme recognition (Waibel et al, 1989), and spoken digit recognition (Franzini et al, 1989). However, in 1990, when this thesis was proposed, it still remained to be seen whether neural networks could support a large vocabulary, speaker independent, continuous speech recognition system.

In this thesis we take an incremental approach to this problem. Of the two types of variability in speech — acoustic and temporal — the former is more naturally posed as a static

pattern matching problem that is amenable to neural networks; therefore we use neural networks for acoustic modeling, while we rely on conventional Hidden Markov Models for temporal modeling. Our research thus represents an exploration of the space of *NN-HMM hybrids*. We explore two different ways to use neural networks for acoustic modeling, namely *prediction* and *classification* of the speech patterns. Prediction is shown to be a weak approach because it lacks discrimination, while classification is shown to be a much stronger approach. We present an extensive series of experiments that we performed to optimize our networks for word recognition accuracy, and show that a properly optimized NN-HMM hybrid system based on classification networks can outperform other systems under similar conditions. Finally, we argue that hybrid NN-HMM systems offer several advantages over pure HMM systems, including better acoustic modeling accuracy, better context sensitivity, more natural discrimination, and a more economical use of parameters.

1.3. Thesis Outline

The first few chapters of this thesis provide some essential background and a summary of related work in speech recognition and neural networks:

- Chapter 2 reviews the field of speech recognition.
- Chapter 3 reviews the field of neural networks.
- Chapter 4 reviews the intersection of these two fields, summarizing both past and present approaches to speech recognition using neural networks.

The remainder of the thesis describes our own research, evaluating both predictive networks and classification networks as acoustic models in NN-HMM hybrid systems:

- Chapter 5 introduces the databases we used in our experiments.
- Chapter 6 presents our research with predictive networks, and explains why this approach yielded poor results.
- Chapter 7 presents our research with classification networks, and shows how we achieved excellent results through an extensive series of optimizations.
- Chapter 8 compares the performance of our optimized systems against many other systems on the same databases, demonstrating the value of NN-HMM hybrids.
- Chapter 9 presents the conclusions of this thesis.

2. Review of Speech Recognition

In this chapter we will present a brief review of the field of speech recognition. After reviewing some fundamental concepts, we will explain the standard Dynamic Time Warping algorithm, and then discuss Hidden Markov Models in some detail, offering a summary of the algorithms, variations, and limitations that are associated with this dominant technology.

2.1. Fundamentals of Speech Recognition

Speech recognition is a multileveled pattern recognition task, in which acoustical signals are examined and structured into a hierarchy of subword units (e.g., phonemes), words, phrases, and sentences. Each level may provide additional temporal constraints, e.g., known word pronunciations or legal word sequences, which can compensate for errors or uncertainties at lower levels. This hierarchy of constraints can best be exploited by combining decisions probabilistically at all lower levels, and making discrete decisions only at the highest level.

The structure of a standard speech recognition system is illustrated in Figure 2.1. The elements are as follows:

- **Raw speech**. Speech is typically sampled at a high frequency, e.g., 16 KHz over a microphone or 8 KHz over a telephone. This yields a sequence of amplitude values over time.
- **Signal analysis.** Raw speech should be initially transformed and compressed, in order to simplify subsequent processing. Many signal analysis techniques are available which can extract useful features and compress the data by a factor of ten without losing any important information. Among the most popular:
    - Fourier analysis (FFT) yields discrete frequencies over time, which can be interpreted visually. Frequencies are often distributed using a *Mel* scale, which is linear in the low range but logarithmic in the high range, corresponding to physiological characteristics of the human ear.
    - Perceptual Linear Prediction (PLP) is also physiologically motivated, but yields coefficients that cannot be interpreted visually.

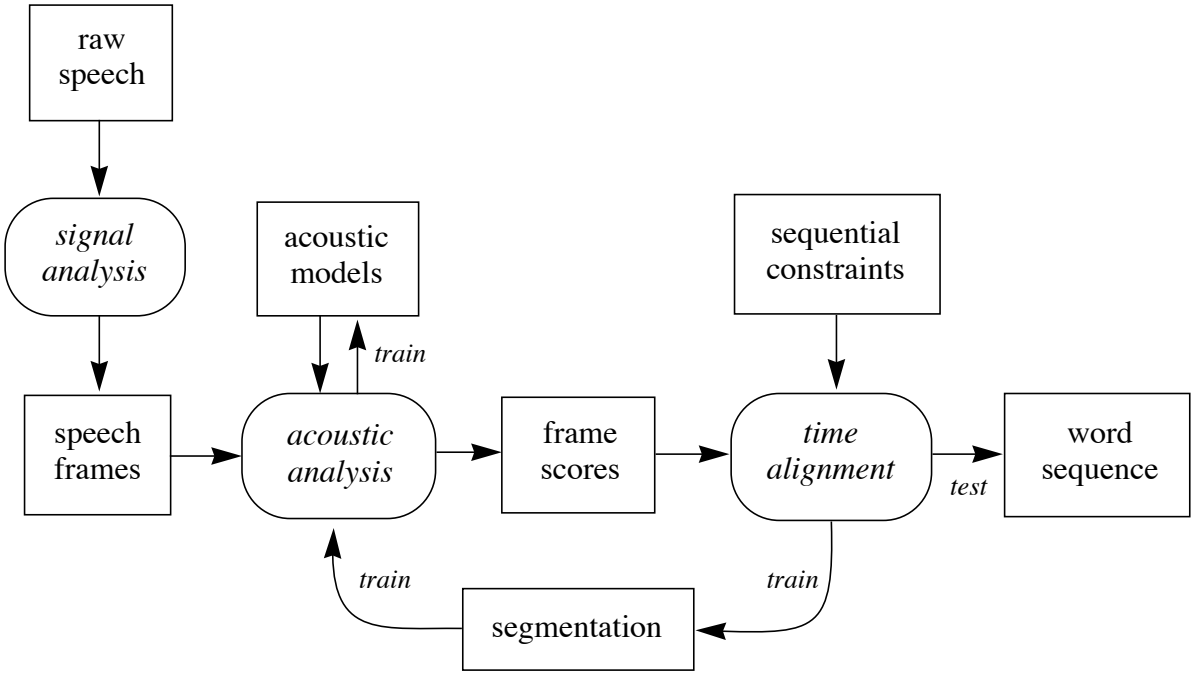


Figure 2.1: Structure of a standard speech recognition system.

- Linear Predictive Coding (LPC) yields coefficients of a linear equation that approximate the recent history of the raw speech values.
- Cepstral analysis calculates the inverse Fourier transform of the logarithm of the power spectrum of the signal.

In practice, it makes little difference which technique is used[1]. Afterwards, procedures such as Linear Discriminant Analysis (LDA) may optionally be applied to further reduce the dimensionality of any representation, and to decorrelate the coefficients.

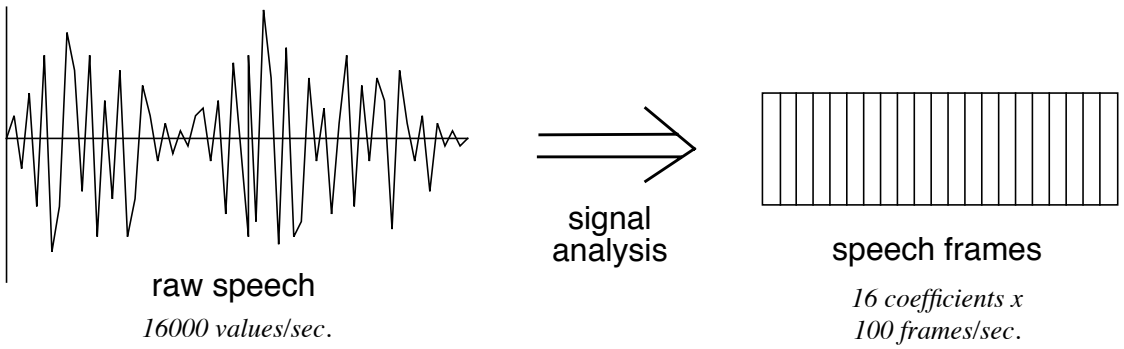


Figure 2.2: Signal analysis converts raw speech to speech frames.

1. Assuming benign conditions. Of course, each technique has its own advocates.

- **Speech frames.** The result of signal analysis is a sequence of *speech frames*, typically at 10 msec intervals, with about 16 coefficients per frame. These frames may be augmented by their own first and/or second derivatives, providing explicit information about speech dynamics; this typically leads to improved performance. The speech frames are used for acoustic analysis.
- **Acoustic models.** In order to analyze the speech frames for their acoustic content, we need a set of *acoustic models*. There are many kinds of acoustic models, varying in their representation, granularity, context dependence, and other properties.

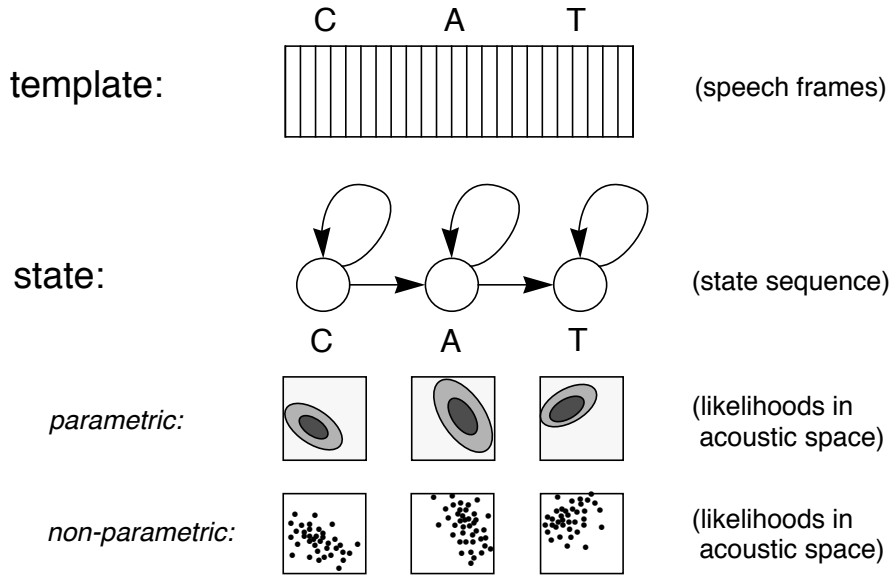


Figure 2.3: Acoustic models: template and state representations for the word "cat".

Figure 2.3 shows two popular representations for acoustic models. The simplest is a *template*, which is just a stored sample of the unit of speech to be modeled, e.g., a recording of a word. An unknown word can be recognized by simply comparing it against all known templates, and finding the closest match. Templates have two major drawbacks: (1) they cannot model acoustic variabilities, except in a coarse way by assigning multiple templates to each word; and (2) in practice they are limited to whole-word models, because it's hard to record or segment a sample shorter than a word — so templates are useful only in small systems which can afford the luxury of using whole-word models. A more flexible representation, used in larger systems, is based on trained acoustic models, or *states*. In this approach, every word is modeled by a sequence of trainable states, and each state indicates the sounds that are likely to be heard in that segment of the word, using a probability distribution over the acoustic space. Probability distributions can be modeled *parametrically*, by assuming that they have a simple shape (e.g., a Gaussian distribution) and then trying to find the parameters that describe it; or *non-parametrically*, by representing the distribution directly (e.g., with a histogram over a quantization of the acoustic space, or, as we shall see, with a neural network).

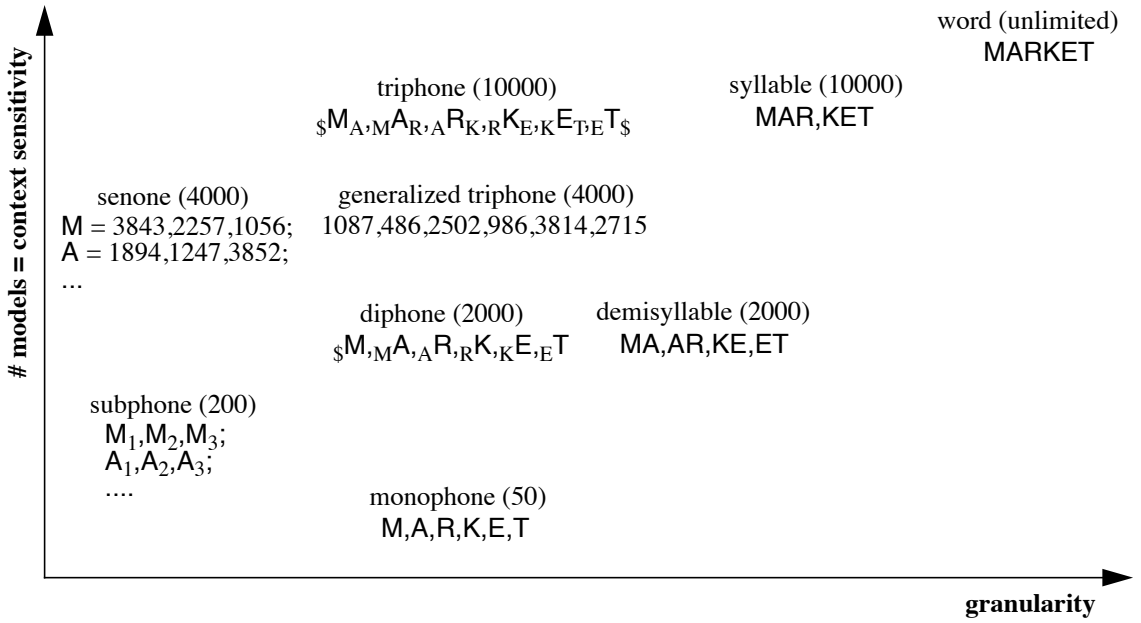


Figure 2.4: Acoustic models: granularity vs. context sensitivity, illustrated for the word "market".

Acoustic models also vary widely in their granularity and context sensitivity. Figure 2.4 shows a chart of some common types of acoustic models, and where they lie along these dimensions. As can be seen, models with larger granularity (such as *word* or *syllable* models) tend to have greater context sensitivity. Moreover, models with the greatest context sensitivity give the best word recognition accuracy —if those models are well trained. Unfortunately, the larger the granularity of a model, the poorer it will be trained, because fewer samples will be available for training it. For this reason, word and syllable models are rarely used in high-performance systems; much more common are *triphone* or *generalized triphone* models. Many systems also use *monophone* models (sometimes simply called *phoneme* models), because of their relative simplicity.

During training, the acoustic models are incrementally modified in order to optimize the overall performance of the system. During testing, the acoustic models are left unchanged.

- **Acoustic analysis and frame scores.** *Acoustic analysis* is performed by applying each acoustic model over each frame of speech, yielding a matrix of *frame scores*, as shown in Figure 2.5. Scores are computed according to the type of acoustic model that is being used. For template-based acoustic models, a score is typically the Euclidean distance between a template's frame and an unknown frame. For state-based acoustic models, a score represents an *emission probability*, i.e., the likelihood of the current state generating the current frame, as determined by the state's parametric or non-parametric function.
- **Time alignment.** Frame scores are converted to a word sequence by identifying a sequence of acoustic models, representing a valid word sequence, which gives the

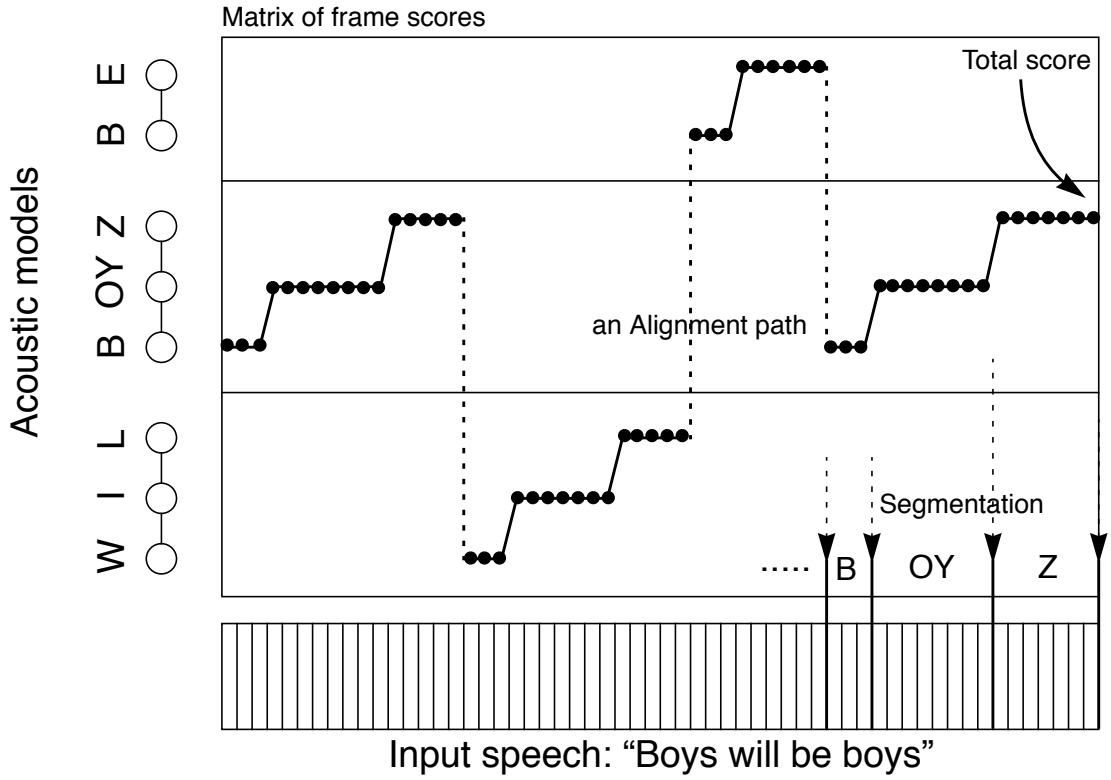


Figure 2.5: The alignment path with the best total score identifies the word sequence and segmentation.

best total score along an *alignment path* through the matrix[1], as illustrated in Figure 2.5. The process of searching for the best alignment path is called *time alignment*.

An alignment path must obey certain *sequential constraints* which reflect the fact that speech always goes forward, never backwards. These constraints are manifested both within and between words. Within a word, sequential constraints are implied by the sequence of frames (for template-based models), or by the sequence of states (for state-based models) that comprise the word, as dictated by the phonetic pronunciations in a dictionary, for example. Between words, sequential constraints are given by a grammar, indicating what words may follow what other words.

Time alignment can be performed efficiently by *dynamic programming*, a general algorithm which uses only local path constraints, and which has linear time and space requirements. (This general algorithm has two main variants, known as *Dynamic Time Warping* (DTW) and *Viterbi search*, which differ slightly in their local computations and in their optimality criteria.)

In a state-based system, the optimal alignment path induces a *segmentation* on the word sequence, as it indicates which frames are associated with each state. This

1. Actually, it is often better to evaluate a state sequence not by its single best alignment path, but by the composite score of all of its possible alignment paths; but we will ignore that issue for now.

segmentation can be used to generate labels for recursively training the acoustic models on corresponding frames.

- **Word sequence.** The end result of time alignment is a *word sequence* — the sentence hypothesis for the utterance. Actually it is common to return several such sequences, namely the ones with the highest scores, using a variation of time alignment called *N-best search* (Schwartz and Chow, 1990). This allows a recognition system to make two passes through the unknown utterance: the first pass can use simplified models in order to quickly generate an N-best list, and the second pass can use more complex models in order to carefully rescore each of the N hypotheses, and return the single best hypothesis.

2.2. Dynamic Time Warping

In this section we motivate and explain the *Dynamic Time Warping* algorithm, one of the oldest and most important algorithms in speech recognition (Vintsyuk 1971, Itakura 1975, Sakoe and Chiba 1978).

The simplest way to recognize an *isolated* word sample is to compare it against a number of stored word templates and determine which is the "best match". This goal is complicated by a number of factors. First, different samples of a given word will have somewhat different durations. This problem can be eliminated by simply normalizing the templates and the unknown speech so that they all have an equal duration. However, another problem is that the rate of speech may not be constant throughout the word; in other words, the optimal alignment between a template and the speech sample may be nonlinear. Dynamic Time Warping (DTW) is an efficient method for finding this optimal nonlinear alignment.

DTW is an instance of the general class of algorithms known as *dynamic programming*. Its time and space complexity is merely linear in the duration of the speech sample and the vocabulary size. The algorithm makes a single pass through a matrix of frame scores while computing locally optimized segments of the global alignment path. (See Figure 2.6.) If $D(x,y)$ is the Euclidean distance between frame x of the speech sample and frame y of the reference template, and if $C(x,y)$ is the cumulative score along an optimal alignment path that leads to (x,y), then

$$C(x, y) = \text{MIN}(C(x-1, y), C(x-1, y-1), C(x, y-1)) + D(x, y) \quad (1)$$

The resulting alignment path may be visualized as a low valley of Euclidean distance scores, meandering through the hilly landscape of the matrix, beginning at $(0, 0)$ and ending at the final point (X, Y). By keeping track of backpointers, the full alignment path can be recovered by tracing backwards from (X, Y). An optimal alignment path is computed for each reference word template, and the one with the lowest cumulative score is considered to be the best match for the unknown speech sample.

There are many variations on the DTW algorithm. For example, it is common to vary the local path constraints, e.g., by introducing transitions with slope 1/2 or 2, or weighting the

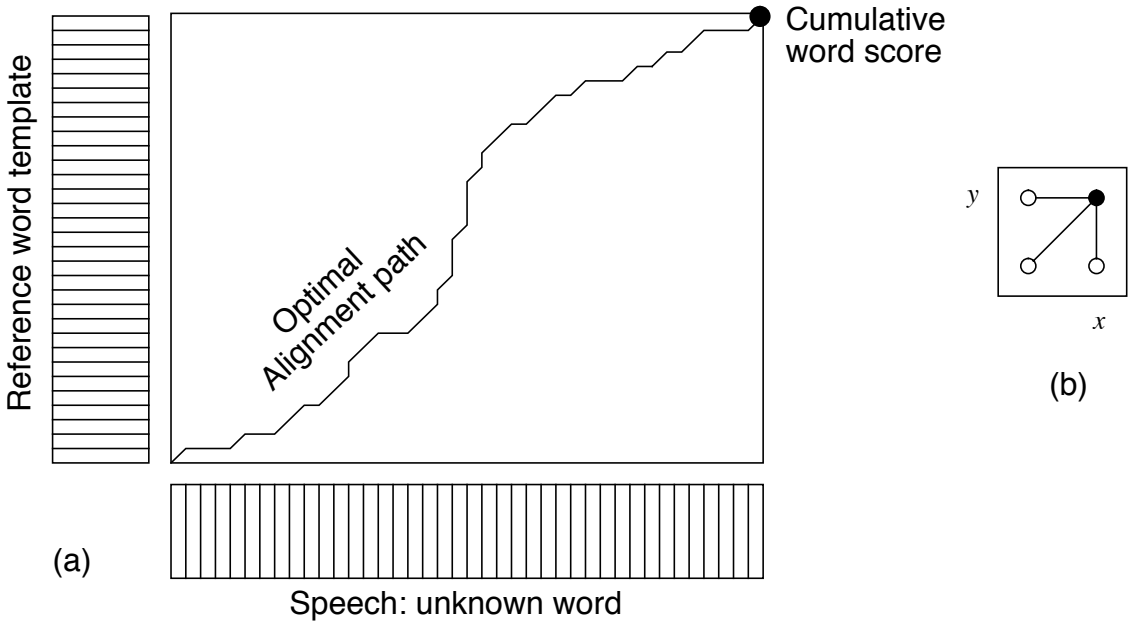


Figure 2.6: Dynamic Time Warping. (a) alignment path. (b) local path constraints.

transitions in various ways, or applying other kinds of slope constraints (Sakoe and Chiba 1978). While the reference word models are usually templates, they may be state-based models (as shown previously in Figure 2.5). When using states, vertical transitions are often disallowed (since there are fewer states than frames), and often the goal is to maximize the cumulative score, rather than to minimize it.

A particularly important variation of DTW is an extension from isolated to continuous speech. This extension is called the *One Stage DTW* algorithm (Ney 1984). Here the goal is to find the optimal alignment between the speech sample and the best sequence of reference words (see Figure 2.5). The complexity of the extended algorithm is still linear in the length of the sample and the vocabulary size. The only modification to the basic DTW algorithm is that at the beginning of each reference word model (i.e., its first frame or state), the diagonal path is allowed to point back to the end of all reference word models in the preceding frame. Local backpointers must specify the reference word model of the preceding point, so that the optimal word sequence can be recovered by tracing backwards from the final point (W, X, Y) of the word W with the best final score. Grammars can be imposed on continuous speech recognition by restricting the allowed transitions at word boundaries.

2.3. Hidden Markov Models

The most flexible and successful approach to speech recognition so far has been Hidden Markov Models (HMMs). In this section we will present the basic concepts of HMMs, describe the algorithms for training and using them, discuss some common variations, and review the problems associated with HMMs.

2.3.1. Basic Concepts

A Hidden Markov Model is a collection of states connected by transitions, as illustrated in Figure 2.7. It begins in a designated initial state. In each discrete time step, a transition is taken into a new state, and then one output symbol is generated in that state. The choice of transition and output symbol are both random, governed by probability distributions. The HMM can be thought of as a black box, where the sequence of output symbols generated over time is observable, but the sequence of states visited over time is hidden from view. This is why it's called a *Hidden* Markov Model.

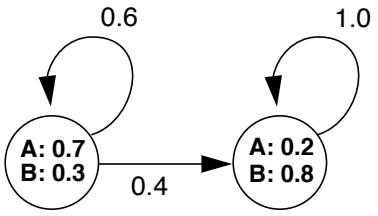


Figure 2.7: A simple Hidden Markov Model, with two states and two output symbols, A and B.

HMMs have a variety of applications. When an HMM is applied to speech recognition, the states are interpreted as acoustic models, indicating what sounds are likely to be heard during their corresponding segments of speech; while the transitions provide temporal constraints, indicating how the states may follow each other in sequence. Because speech always goes forward in time, transitions in a speech application always go forward (or make a self-loop, allowing a state to have arbitrary duration). Figure 2.8 illustrates how states and transitions in an HMM can be structured hierarchically, in order to represent phonemes, words, and sentences.

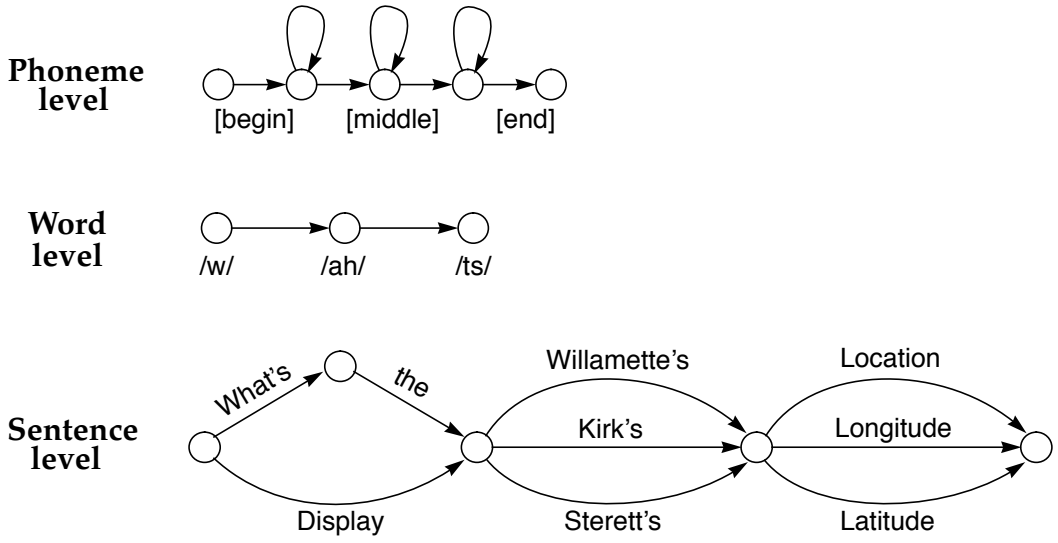


Figure 2.8: A hierarchically structured HMM.

Formally, an HMM consists of the following elements:

$\{s\}$ = A set of states.

$\{a_{ij}\}$ = A set of transition probabilities, where a_{ij} is the probability of taking the transition from state i to state j.

$\{b_i(u)\}$ = A set of emission probabilities, where b_i is the probability distribution over the acoustic space describing the likelihood of emitting[1] each possible sound u while in state i.

Since a and b are both probabilities, they must satisfy the following properties:

$$a_{ij} \geq 0, \quad b_i(u) \geq 0, \quad \forall i,j,u \tag{2}$$

$$\sum_j a_{ij} = 1, \quad \forall i \tag{3}$$

$$\sum_u b_i(u) = 1, \quad \forall i \tag{4}$$

In using this notation we implicitly confine our attention to First-Order HMMs, in which $\boldsymbol{a}$ and $\boldsymbol{b}$ depend only on the current state, independent of the previous history of the state sequence. This assumption, almost universally observed, limits the number of trainable parameters and makes the training and testing algorithms very efficient, rendering HMMs useful for speech recognition.

2.3.2. Algorithms

There are three basic algorithms associated with Hidden Markov Models:

- the *forward algorithm*, useful for isolated word recognition;
- the *Viterbi algorithm*, useful for continuous speech recognition; and
- the *forward-backward algorithm*, useful for training an HMM.

In this section we will review each of these algorithms.

2.3.2.1. The Forward Algorithm

In order to perform isolated word recognition, we must be able to evaluate the probability that a given HMM word model produced a given observation sequence, so that we can compare the scores for each word model and choose the one with the highest score. More formally: given an HMM model $\boldsymbol{M}$, consisting of $\{s\}$, $\{a_{ij}\}$, and $\{b_i(u)\}$, we must compute the probability that it generated the output sequence $y_1^T = (y_1, y_2, y_3, ..., y_T)$. Because every state i can generate each output symbol u with probability $b_i(u)$, every state sequence of length T

1. It is traditional to refer to $b_i(u)$ as an "emission" probability rather than an "observation" probability, because an HMM is traditionally a generative model, even though we are using it for speech recognition. The difference is moot.

contributes something to the total probability. A brute force algorithm would simply list all possible state sequences of length T, and accumulate their probabilities of generating y_1^T; but this is clearly an exponential algorithm, and is not practical.

A much more efficient solution is the *Forward Algorithm*, which is an instance of the class of algorithms known as *dynamic programming*, requiring computation and storage that are only linear in T. First, we define $\alpha_j(t)$ as the probability of generating the partial sequence y_1^t, ending up in state j at time t. $\alpha_j(t=0)$ is initialized to 1.0 in the initial state, and 0.0 in all other states. If we have already computed $\alpha_i(t-1)$ for all i in the previous time frame $t-1$, then $\alpha_j(t)$ can be computed recursively in terms of the incremental probability of entering state j from each i while generating the output symbol y_t (see Figure 2.9):

$$\alpha_j(t) = \sum_i \alpha_i(t-1)\, a_{ij} b_j(y_t) \tag{5}$$

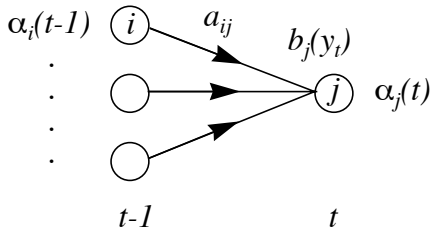


Figure 2.9: The forward pass recursion.

If F is the final state, then by induction we see that $\alpha_F(T)$ is the probability that the HMM generated the complete output sequence y_1^T.

Figure 2.10 shows an example of this algorithm in operation, computing the probability that the output sequence y_1^3 =`(A,A,B)` could have been generated by the simple HMM presented earlier. Each cell at (t,j) shows the value of $\alpha_j(t)$, using the given values of $\boldsymbol{a}$ and $\boldsymbol{b}$. The computation proceeds from the first state to the last state within a time frame, before proceeding to the next time frame. In the final cell, we see that the probability that this particular HMM generates the sequence `(A,A,B)` is .096.

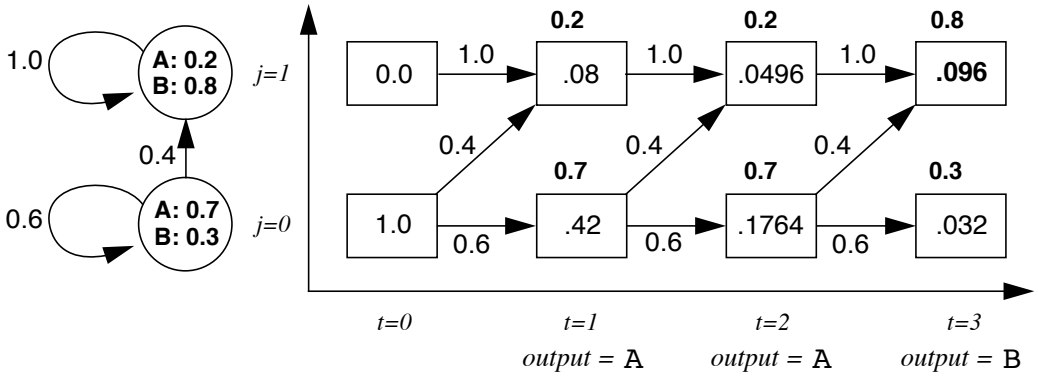


Figure 2.10: An illustration of the forward algorithm, showing the value of $\alpha_j(t)$ in each cell.

2.3.2.2. The Viterbi Algorithm

While the Forward Algorithm is useful for isolated word recognition, it cannot be applied to continuous speech recognition, because it is impractical to have a separate HMM for each possible sentence. In order to perform continuous speech recognition, we should instead infer the actual sequence of states that generated the given observation sequence; from the state sequence we can easily recover the word sequence. Unfortunately the actual state sequence is hidden (by definition), and cannot be uniquely identified; after all, any path could have produced this output sequence, with some small probability. The best we can do is to find the *one* state sequence that was *most likely* to have generated the observation sequence. As before, we could do this by evaluating all possible state sequences and reporting the one with the highest probability, but this would be an exponential and hence infeasible algorithm.

A much more efficient solution is the *Viterbi Algorithm*, which is again based on dynamic programming. It is very similar to the Forward Algorithm, the main difference being that instead of evaluating a summation at each cell, we evaluate the maximum:

$$v_j(t) = MAX_i\left[v_i(t-1)\, a_{ij} b_j(y_t) \right] \tag{6}$$

This implicitly identifies the single best predecessor state for each cell in the matrix. If we explicitly identify that best predecessor state, saving a single backpointer in each cell in the matrix, then by the time we have evaluated $v_F(T)$ at the final state at the final time frame, we can retrace those backpointers from the final cell to reconstruct the whole state sequence. Figure 2.11 illustrates this process. Once we have the state sequence (i.e., an alignment path), we can trivially recover the word sequence.

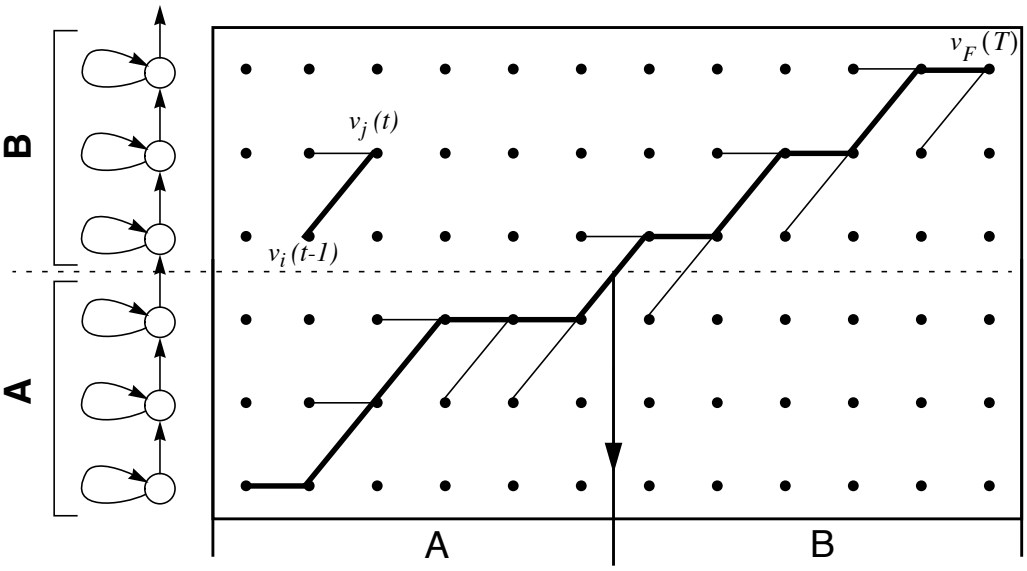


Figure 2.11: An example of backtracing.

2.3.2.3. The Forward-Backward Algorithm

In order to train an HMM, we must optimize ***a*** and ***b*** with respect to the HMM's likelihood of generating all of the output sequences in the training set, because this will maximize the HMM's chances of also correctly recognizing new data. Unfortunately this is a difficult problem; it has no closed form solution. The best that can be done is to start with some initial values for ***a*** and ***b***, and then to iteratively modify ***a*** and ***b*** by reestimating and improving them, until some stopping criterion is reached. This general method is called *Estimation-Maximization* (EM). A popular instance of this general method is the *Forward-Backward Algorithm* (also known as the *Baum-Welch Algorithm*), which we now describe.

Previously we defined $\alpha_j(t)$ as the probability of generating the partial sequence y_1^t and ending up in state j at time t. Now we define its mirror image, $\beta_j(t)$, as the probability of generating the remainder of the sequence y_{t+1}^T, starting from state j at time t. $\alpha_j(t)$ is called the *forward term*, while $\beta_j(t)$ is called the *backward term*. Like $\alpha_j(t)$, $\beta_j(t)$ can be computed recursively, but this time in a backward direction (see Figure 2.12):

$$\beta_j(t) = \sum_k a_{jk} b_k(y_{t+1}) \beta_k(t+1) \tag{7}$$

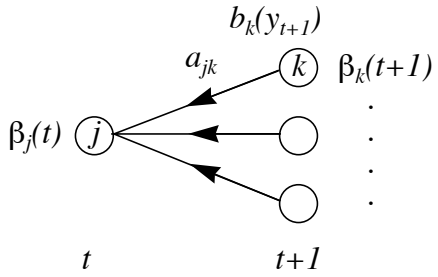


Figure 2.12: The backward pass recursion.

This recursion is initialized at time T by setting $\beta_k(T)$ to 1.0 for the final state, and 0.0 for all other states.

Now we define $\gamma_{ij}(t)$ as the probability of transitioning from state i to state j at time t, given that the whole output sequence y_1^T has been generated by the current HMM:

$$\gamma_{ij}(t) = P(i_t \rightarrow j | y_1^T) = \frac{P(i_t \rightarrow j, y_1^T)}{P(y_1^T)} = \frac{\alpha_i(t) a_{ij} b_j(y_{t+1}) \beta_j(t+1)}{\sum_k \alpha_k(T)} \tag{8}$$

The numerator in the final equality can be understood by consulting Figure 2.13. The denominator reflects the fact that the probability of generating y_1^T equals the probability of generating y_1^T while ending up in any of k final states.

Now let us define $N(i \rightarrow j)$ as the expected number of times that the transition from state i to state j is taken, from time 1 to T:

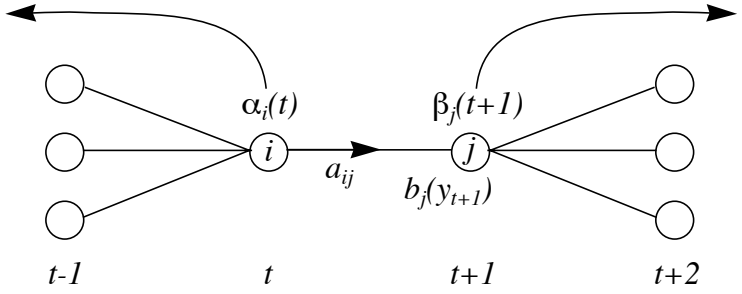


Figure 2.13: Deriving $\gamma_{ij}(t)$ in the Forward-Backward Algorithm.

$$N(i \rightarrow j) = \sum_t \gamma_{ij}(t) \tag{9}$$

Summing this over all destination states j, we obtain $N(i \rightarrow *)$, or $N(i)$, which represents the expected number of times that state i is visited, from time 1 to T:

$$N(i) = N(i \rightarrow *) = \sum_j \sum_t \gamma_{ij}(t) \tag{10}$$

Selecting only those occasions when state i emits the symbol u, we obtain $N(i, u)$:

$$N(i, u) = \sum_{t:(y_t=u)} \sum_j \gamma_{ij}(t) \tag{11}$$

Finally, we can reestimate the HMM parameters $\boldsymbol{a}$ and $\boldsymbol{b}$, yielding $\overline{\boldsymbol{a}}$ and $\overline{\boldsymbol{b}}$, by taking simple ratios between these terms:

$$\overline{a_{ij}} = P(i \rightarrow j) = \frac{N(i \rightarrow j)}{N(i \rightarrow *)} = \frac{\sum_t \gamma_{ij}(t)}{\sum_j \sum_t \gamma_{ij}(t)} \tag{12}$$

$$\overline{b_i(u)} = P(i, u) = \frac{N(i, u)}{N(i)} = \frac{\sum_{t:(y_t=u)} \sum_j \gamma_{ij}(t)}{\sum_t \sum_j \gamma_{ij}(t)} \tag{13}$$

It can be proven that substituting $\{\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}\}$ for $\{\boldsymbol{a}, \boldsymbol{b}\}$ will always cause $P(y_1^T)$ to increase, up to a local maximum. Thus, by repeating this procedure for a number of iterations, the HMM parameters will be optimized for the training data, and will hopefully generalize well to testing data.

2.3.3. Variations

There are many variations on the standard HMM model. In this section we discuss some of the more important variations.

2.3.3.1. Density Models

The states of an HMM need some way to model probability distributions in acoustic space. There are three popular ways to do this, as illustrated in Figure 2.14:

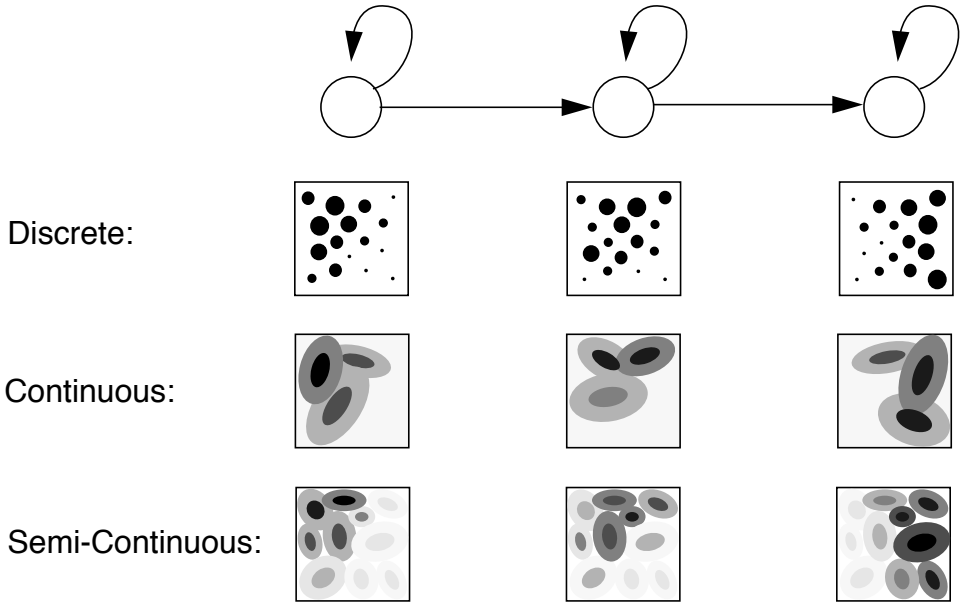


Figure 2.14: Density models, describing the probability density in acoustic space.

- **Discrete density model** (Lee 1988). In this approach, the entire acoustic space is divided into a moderate number (e.g., 256) of regions, by a clustering procedure known as Vector Quantization (VQ). The centroid of each cluster is represented by a scalar codeword, which is an index into a codebook that identifies the corresponding acoustic vectors. Each input frame is converted to a codeword by finding the nearest vector in the codebook. The HMM output symbols are also codewords. Thus, the probability distribution over acoustic space is represented by a simple histogram over the codebook entries. The drawback of this nonparametric approach is that it suffers from quantization errors if the codebook is too small, while increasing the codebook size would leave less training data for each codeword, likewise degrading performance.
- **Continuous density model** (Woodland et al, 1994). Quantization errors can be eliminated by using a continuous density model, instead of VQ codebooks. In this approach, the probability distribution over acoustic space is modeled directly, by assuming that it has a certain parametric form, and then trying to find those param-

eters. Typically this parametric form is taken to be a mixture of K Gaussians, i.e.,

$$b_j(y) = \sum_{k=1}^{K} c_{jk} G(y, \mu_{jk}, U_{jk}) \tag{14}$$

where c_{jk} is the weighting factor for each Gaussian G with mean μ_{jk} and covariance matrix U_{jk}, such that $\sum_k c_{jk} = 1$. During training, the reestimation of $\boldsymbol{b}$ then involves the reestimation of c_{jk}, μ_{jk}, and U_{jk}, using an additional set of formulas. The drawback of this approach is that parameters are not shared between states, so if there are many states in the whole system, then a large value of K may yield too many total parameters to be trained adequately, while decreasing the value of K may invalidate the assumption that the distribution can be well-modeled by a mixture of Gaussians.

- **Semi-Continuous density model** (Huang 1992), also called the **Tied-Mixture model** (Bellagarda and Nahamoo 1988). This is a compromise between the above two approaches. In a Semi-Continuous density model, as in the discrete model, there is a codebook describing acoustic clusters, shared by all states. But rather than representing the clusters as discrete centroids to which nearby vectors are collapsed, they are represented as continuous density functions (typically Gaussians) over the neighboring space, thus avoiding quantization errors. That is,

$$b_j(y) = \sum_{k=1}^{L} c_{jk} G(y, \mu_k, U_k) \tag{15}$$

where L is the number of codebook entries, and c_{jk} is the weighting factor for each Gaussian G with mean μ_k and covariance matrix U_k. As in the continuous case, the Gaussians are reestimated during training, hence the codebook is optimized jointly with the HMM parameters, in contrast to the discrete model in which the codebook remains fixed. This joint optimization can further improve the system's performance.

All three density models are widely used, although continuous densities seem to give the best results on large databases (while running up to 300 times slower, however).

2.3.3.2. Multiple Data Streams

So far we have discussed HMMs that assume a single data stream, i.e., input acoustic vectors. HMMs can be modified to use multiple streams, such that

$$b_j(u) = \prod_{i=1}^{N} b_j(u_i) \tag{16}$$

where u_i are the observation vectors of N independent data streams, which are modeled with separate codebooks or Gaussian mixtures. HMM based speech recognizers commonly[1] use up to four data streams, for example representing spectral coefficients, delta spectral coeffi-

cients, power, and delta power. While it is possible to concatenate each of these into one long vector, and to vector-quantize that single data stream, it is generally better to treat these separate data streams independently, so that each stream is more coherent and their union can be modeled with a minimum of parameters.

2.3.3.3. Duration modeling

If the self-transition probability $a_{ii} = p$, then the probability of remaining in state i for d frames is p^d, indicating that state duration in an HMM is modeled by exponential decay. Unfortunately this is a poor model of duration, as state durations actually have a roughly Poisson distribution. There are several ways to improve duration modeling in HMMs.

We can define $p_i(d)$ as the probability of remaining in state i for a duration of d frames, and create a histogram of $p_i(d)$ from the training data. To ensure that state duration is governed by $p_i(d)$, we must eliminate all self-loops (by setting a_{ii}=0), and modify the equations for α and β as well as all the reestimation formulas, to include summations over d (up to a maximum duration D) of terms with multiplicative factors that represent all possible durational contingencies. Unfortunately this increases memory requirements by a factor of D, and computational requirements by a factor of $D^2/2$. If D=25 frames (which is quite reasonable), this causes the application to run about 300 times slower. Another problem with this approach is that it may require more training parameters (adding about 25 per state) than can be adequately trained.

The latter problem can be mitigated by replacing the above nonparametric approach with a parametric approach, in which a Poisson, Gaussian, or Gamma distribution is assumed as a duration model, so that relatively few parameters are needed. However, this improvement causes the system to run even slower.

A third possibility is to ignore the precise shape of the distribution, and simply impose hard minimum and maximum duration constraints. One way to impose these constraints is by duplicating the states and modifying the state transitions appropriately. This approach has only moderate overhead, and gives fairly good results, so it tends to be the most favored approach to duration modeling.

2.3.3.4. Optimization criteria

The training procedure described earlier (the Forward-Backward Algorithm) implicitly uses an optimization criterion known as *Maximum Likelihood* (ML), which maximizes the likelihood that a given observation sequence Y is generated by the correct model M_c, without considering other models M_i. (For instance, if M_i represent word models, then only the correct word model will be updated with respect to Y, while all the competing word models are ignored.) Mathematically, ML training solves for the HMM parameters $\Lambda = \{\boldsymbol{a}, \boldsymbol{b}\}$, and specifically the subset Λ_c that corresponds to the correct model M_c, such that

$$\Lambda_{ML} = \underset{\Lambda}{\operatorname{argmax}}\ P(Y|\Lambda_c) \tag{17}$$

1. Although this is still common among semi-continuous HMMs, there is now a trend towards using a single data stream with LDA coefficients derived from these separate streams; this latter approach is now common among continuous HMMs.

If the HMM's modeling assumptions were accurate — e.g., if the probability density in acoustic space could be precisely modeled by a mixture of Gaussians, and if enough training data were available for perfectly estimating the distributions — then ML training would theoretically yield optimal recognition accuracy. But the modeling assumptions are always inaccurate, because acoustic space has a complex terrain, training data is limited, and the scarcity of training data limits the size and power of the models, so that they cannot perfectly fit the distributions. This unfortunate condition is called *model mismatch*. An important consequence is that ML is not guaranteed to be the optimal criterion for training an HMM.

An alternative criterion is *Maximum Mutual Information* (MMI), which enhances discrimination between competing models, in an attempt to squeeze as much useful information as possible out of the limited training data. In this approach, the correct model M_c is trained positively while all other models M_i are trained negatively on the observation sequence Y, helping to separate the models and improve their ability to discriminate during testing. Mutual information between an observation sequence Y and the correct model M_c is defined as follows:

$$\begin{aligned} I_\Lambda(Y, M_c) &= \log \frac{P(Y, M_c)}{P(Y)\,P(M_c)} = \log \frac{P(Y|M_c)}{P(Y)} = \log P(Y|M_c) - \log P(Y) \\ &= \log P(Y|M_c) - \log \sum_i P(Y|M_i)\,P(M_i) \end{aligned} \tag{18}$$

where the first term represents positive training on the correct model M_c (just as in ML), while the second term represents negative training on all other models M_i. Training with the MMI criterion then involves solving for the model parameters Λ that maximize the mutual information:

$$\Lambda_{MMI} = \operatorname*{argmax}_\Lambda I_\Lambda(Y, M_c) \tag{19}$$

Unfortunately, this equation cannot be solved by either direct analysis or reestimation; the only known way to solve it is by gradient descent, and the proper implementation is complex (Brown 1987, Rabiner 1989).

We note in passing that MMI is equivalent to using a *Maximum A Posteriori* (MAP) criterion, in which the expression to be maximized is $P(M_c|Y)$, rather than $P(Y|M_c)$. To see this, note that according to Bayes Rule,

$$P(M_c|Y) = \frac{P(Y|M_c)\,P(M_c)}{P(Y)} \tag{20}$$

Maximizing this expression is equivalent to maximizing $I_\Lambda(Y, M_c)$, because the distinguishing logarithm is monotonic and hence transparent, and the MAP's extra factor of $P(M_c)$ is transparent because it's only an additive constant (after taking logarithms), whose value is fixed by the HMM's topology and language model.

2.3.4. Limitations of HMMs

Despite their state-of-the-art performance, HMMs are handicapped by several well-known weaknesses, namely:

- The First-Order Assumption — which says that all probabilities depend solely on the current state — is false for speech applications. One consequence is that HMMs have difficulty modeling coarticulation, because acoustic distributions are in fact strongly affected by recent state history. Another consequence is that durations are modeled inaccurately by an exponentially decaying distribution, rather than by a more accurate Poisson or other bell-shaped distribution.
- The Independence Assumption — which says that there is no correlation between adjacent input frames — is also false for speech applications. In accordance with this assumption, HMMs examine only one frame of speech at a time. In order to benefit from the context of neighboring frames, HMMs must 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).
- The HMM probability density models (discrete, continuous, and semi-continuous) have suboptimal modeling accuracy. Specifically, discrete density HMMs suffer from quantization errors, while continuous or semi-continuous density HMMs suffer from model mismatch, i.e., a poor match between their a priori choice of statistical model (e.g., a mixture of K Gaussians) and the true density of acoustic space.
- The Maximum Likelihood training criterion leads to poor discrimination between the acoustic models (given limited training data and correspondingly limited models). Discrimination can be improved using the Maximum Mutual Information training criterion, but this is more complex and difficult to implement properly.

Because HMMs suffer from all these weaknesses, they can obtain good performance only by relying on context dependent phone models, which have so many parameters that they must be extensively shared — and this, in turn, calls for elaborate mechanisms such as senones and decision trees (Hwang et al, 1993b).

We will argue that neural networks mitigate each of the above weaknesses (except the First Order Assumption), while they require relatively few parameters, so that a neural network based speech recognition system can get equivalent or better performance with less complexity.

3. Review of Neural Networks

In this chapter we present a brief review of neural networks. After giving some historical background, we will review some fundamental concepts, describe different types of neural networks and training procedures (with special emphasis on backpropagation), and discuss the relationship between neural networks and conventional statistical techniques.

3.1. Historical Development

The modern study of neural networks actually began in the 19th century, when neurobiologists first began extensive studies of the human nervous system. Cajal (1892) determined that the nervous system is comprised of discrete neurons, which communicate with each other by sending electrical signals down their long *axons*, which ultimately branch out and touch the *dendrites* (receptive areas) of thousands of other neurons, transmitting the electrical signals through *synapses* (points of contact, with variable resistance). This basic picture was elaborated on in the following decades, as different kinds of neurons were identified, their electrical responses were analyzed, and their patterns of connectivity and the brain's gross functional areas were mapped out. While neurobiologists found it relatively easy to study the functionality of individual neurons (and to map out the brain's gross functional areas), it was extremely difficult to determine how neurons worked together to achieve high-level functionality, such as perception and cognition. With the advent of high-speed computers, however, it finally became possible to build working models of neural systems, allowing researchers to freely experiment with such systems and better understand their properties.

McCulloch and Pitts (1943) proposed the first computational model of a neuron, namely the *binary threshold unit*, whose output was either 0 or 1 depending on whether its net input exceeded a given threshold. This model caused a great deal of excitement, for it was shown that a system of such neurons, assembled into a finite state automaton, could compute any arbitrary function, given suitable values of weights between the neurons (see Minsky 1967). Researchers soon began searching for *learning procedures* that would automatically find the values of weights enabling such a network to compute any specific function. Rosenblatt (1962) discovered an iterative learning procedure for a particular type of network, the *single-layer perceptron*, and he proved that this learning procedure always converged to a set of weights that produced the desired function, as long as the desired function was potentially computable by the network. This discovery caused another great wave of excitement, as many AI researchers imagined that the goal of machine intelligence was within reach.

However, in a rigorous analysis, Minsky and Papert (1969) showed that the set of functions potentially computable by a single-layer perceptron is actually quite limited, and they expressed pessimism about the potential of multi-layer perceptrons as well; as a direct result, funding for connectionist research suddenly dried up, and the field lay dormant for 15 years.

Interest in neural networks was gradually revived when Hopfield (1982) suggested that a network can be analyzed in terms of an *energy function*, triggering the development of the *Boltzmann Machine* (Ackley, Hinton, & Sejnowski 1985) — a stochastic network that could be trained to produce any kind of desired behavior, from arbitrary pattern mapping to pattern completion. Soon thereafter, Rumelhart et al (1986) popularized a much faster learning procedure called *backpropagation*, which could train a multi-layer perceptron to compute any desired function, showing that Minsky and Papert's earlier pessimism was unfounded. With the advent of backpropagation, neural networks have enjoyed a third wave of popularity, and have now found many useful applications.

3.2. Fundamentals of Neural Networks

In this section we will briefly review the fundamentals of neural networks. There are many different types of neural networks, but they all have four basic attributes:

- A set of processing units;
- A set of connections;
- A computing procedure;
- A training procedure.

Let us now discuss each of these attributes.

3.2.1. Processing Units

A neural network contains a potentially huge number of very simple processing units, roughly analogous to neurons in the brain. All these units operate simultaneously, supporting massive parallelism. All computation in the system is performed by these units; there is no other processor that oversees or coordinates their activity[1]. At each moment in time, each unit simply computes a scalar function of its local inputs, and broadcasts the result (called the *activation value*) to its neighboring units.

The units in a network are typically divided into *input units*, which receive data from the environment (such as raw sensory information); *hidden units*, which may internally transform the data representation; and/or *output units*, which represent decisions or control signals (which may control motor responses, for example).

1. Except, of course, to the extent that the neural network may be simulated on a conventional computer, rather than implemented directly in hardware.

In drawings of neural networks, units are usually represented by circles. Also, by convention, input units are usually shown at the bottom, while the outputs are shown at the top, so that processing is seen to be "bottom-up".

The *state* of the network at each moment is represented by the set of activation values over all the units; the network's state typically varies from moment to moment, as the inputs are changed, and/or feedback in the system causes the network to follow a dynamic trajectory through state space.

3.2.2. Connections

The units in a network are organized into a given topology by a set of *connections*, or *weights*, shown as lines in a diagram. Each weight has a real value, typically ranging from $-\infty$ to $+\infty$, although sometimes the range is limited. The value (or *strength*) of a weight describes how much influence a unit has on its neighbor; a positive weight causes one unit to excite another, while a negative weight causes one unit to inhibit another. Weights are usually one-directional (from input units towards output units), but they may be two-directional (especially when there is no distinction between input and output units).

The values of all the weights predetermine the network's computational reaction to any arbitrary input pattern; thus the weights encode the *long-term memory*, or the *knowledge*, of the network. Weights can change as a result of training, but they tend to change slowly, because accumulated knowledge changes slowly. This is in contrast to activation patterns, which are transient functions of the current input, and so are a kind of *short-term memory.*

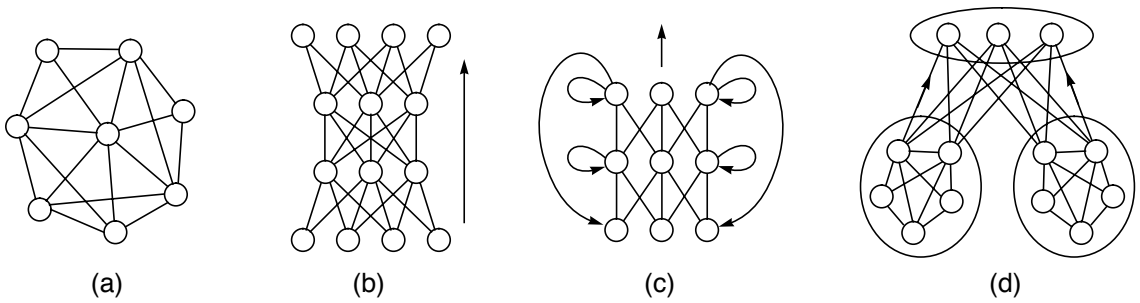


Figure 3.1: Neural network topologies: (a) unstructured, (b) layered, (c) recurrent, (d) modular.

A network can be connected with any kind of topology. Common topologies include unstructured, layered, recurrent, and modular networks, as shown in Figure 3.1. Each kind of topology is best suited to a particular type of application. For example:

- *unstructured networks* are most useful for pattern completion (i.e., retrieving stored patterns by supplying any part of the pattern);
- *layered networks* are useful for pattern association (i.e., mapping input vectors to output vectors);
- *recurrent networks* are useful for pattern sequencing (i.e., following sequences of

network activation over time); and

- *modular networks* are useful for building complex systems from simpler components.

Note that unstructured networks may contain cycles, and hence are actually recurrent; layered networks may or may not be recurrent; and modular networks may integrate different kinds of topologies. In general, unstructured networks use 2-way connections, while other networks use 1-way connections.

Connectivity between two groups of units, such as two layers, is often *complete* (connecting all to all), but it may also be *random* (connecting only some to some), or *local* (connecting one neighborhood to another). A completely connected network has the most degrees of freedom, so it can theoretically learn more functions than more constrained networks; however, this is not always desirable. If a network has too many degrees of freedom, it may simply memorize the training set without learning the underlying structure of the problem, and consequently it may generalize poorly to new data. Limiting the connectivity may help constrain the network to find economical solutions, and so to generalize better. Local connectivity, in particular, can be very helpful when it reflects topological constraints inherent in a problem, such as the geometric constraints that are present between layers in a visual processing system.

3.2.3. Computation

Computation always begins by presenting an input pattern to the network, or *clamping* a pattern of activation on the input units. Then the activations of all of the remaining units are computed, either *synchronously* (all at once in a parallel system) or *asynchronously* (one at a time, in either randomized or natural order), as the case may be. In unstructured networks, this process is called *spreading activation*; in layered networks, it is called *forward propagation*, as it progresses from the input layer to the output layer. In feedforward networks (i.e., networks without feedback), the activations will stabilize as soon as the computations reach the output layer; but in recurrent networks (i.e., networks with feedback), the activations may never stabilize, but may instead follow a dynamic trajectory through state space, as units are continuously updated.

A given unit is typically updated in two stages: first we compute the unit's *net input* (or internal activation), and then we compute its *output activation* as a function of the net input. In the standard case, as shown in Figure 3.2(a), the net input x_j for unit j is just the weighted sum of its inputs:

$$x_j = \sum_i y_i w_{ji} \tag{21}$$

where y_i is the output activation of an incoming unit, and w_{ji} is the weight from unit i to unit j. Certain networks, however, will support so-called *sigma-pi* connections, as shown in Figure 3.2(b), where activations are multiplied together (allowing them to gate each other) before being weighted. In this case, the net input is given by:

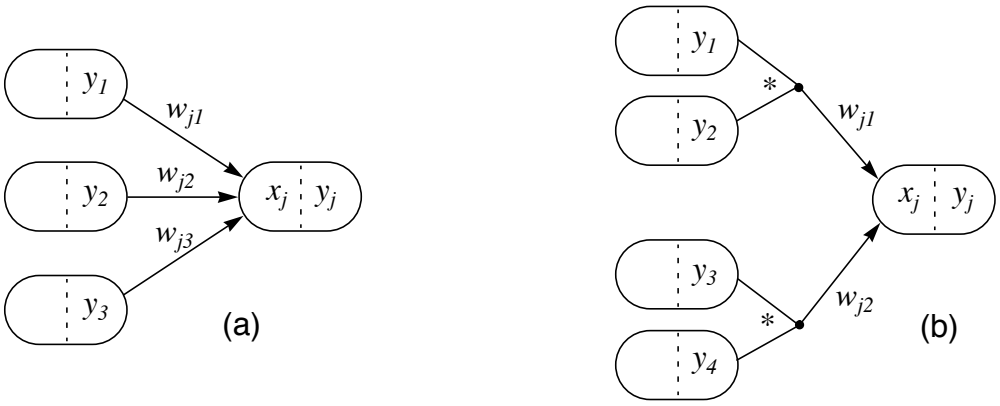


Figure 3.2: Computing unit activations: *x*=net input, *y*=activation. (a) standard unit; (b) sigma-pi unit.

$$x_j = \sum_i w_{ji} \prod_{k \in k(i)} y_k \tag{22}$$

from which the name "sigma-pi" is transparently derived.

In general, the net input is offset by a variable bias term, θ, so that for example Equation (21) is actually:

$$x_j = \sum_i y_i w_{ji} + \theta_j \tag{23}$$

However, in practice, this bias is usually treated as another weight w_{j0} connected to an invisible unit with activation $y_0 = 1$, so that the bias is automatically included in Equation (21) if the summation's range includes this invisible unit.

Once we have computed the unit's net input x_j, we compute the output activation y_j as a function of x_j. This *activation function* (also called a *transfer function*) can be either deterministic or stochastic, and either local or nonlocal.

Deterministic local activation functions usually take one of three forms — *linear, threshold,* or *sigmoidal* — as shown in Figure 3.3. In the linear case, we have simply $y = x$. This is not used very often because it's not very powerful: multiple layers of linear units can be collapsed into a single layer with the same functionality. In order to construct nonlinear functions, a network requires nonlinear units. The simplest form of nonlinearity is provided by the threshold activation function, illustrated in panel (b):

$$y = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{24}$$

This is much more powerful than a linear function, as a multilayered network of threshold units can theoretically compute any boolean function. However, it is difficult to train such a network because the discontinuities in the function imply that finding the desired set of weights may require an exponential search; a practical learning rule exists only for single-

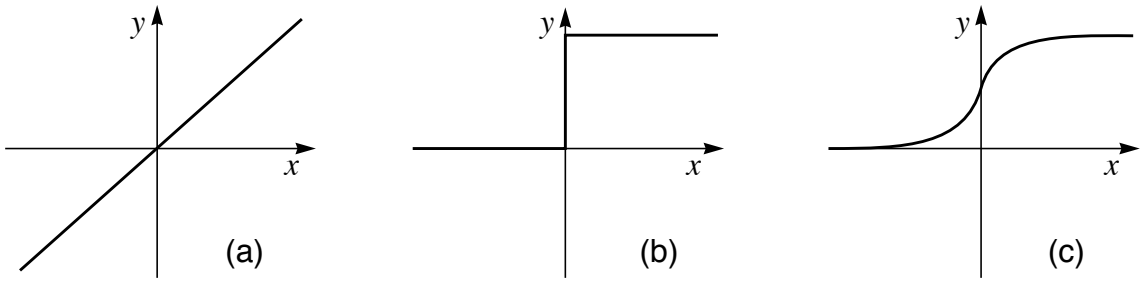


Figure 3.3: Deterministic local activation functions: (a) linear; (b) threshold; (c) sigmoidal.

layered networks of such units, which have limited functionality. Moreover, there are many applications where continuous outputs are preferable to binary outputs. Consequently, the most common function is now the sigmoidal function, illustrated in panel (c):

$$y = \frac{1}{1 + \exp(-x)} \qquad \text{or similarly} \qquad y = \tanh(x) \tag{25}$$

Sigmoidal functions have the advantages of nonlinearity, continuousness, and differentiability, enabling a multilayered network to compute any arbitrary real-valued function, while also supporting a practical training algorithm, backpropagation, based on gradient descent.

Nonlocal activation functions can be useful for imposing global constraints on the network. For example, sometimes it is useful to force all of the network's output activations to sum to 1, like probabilities. This can be performed by linearly normalizing the outputs, but a more popular approach is to use the *softmax* function:

$$y_j = \frac{\exp(x_j)}{\sum_i \exp(x_i)} \tag{26}$$

which operates on the net inputs directly. Nonlocal functions require more overhead and/or hardware, and so are biologically implausible, but they can be useful when global constraints are desired.

Nondeterministic activation functions, in contrast to deterministic ones, are probabilistic in nature. They typically produce binary activation values (0 or 1), where the probability of outputing a 1 is given by:

$$P(y = 1) = \frac{1}{1 + \exp(-x/T)} \tag{27}$$

Here T is a variable called the *temperature*, which commonly varies with time. Figure 3.4 shows how this probability function varies with the temperature: at infinite temperature we have a uniform probability function; at finite temperatures we have sigmoidal probability functions; and at zero temperature we have a binary threshold probability function. If the temperature is steadily decreased during training, in a process called *simulated annealing*, a

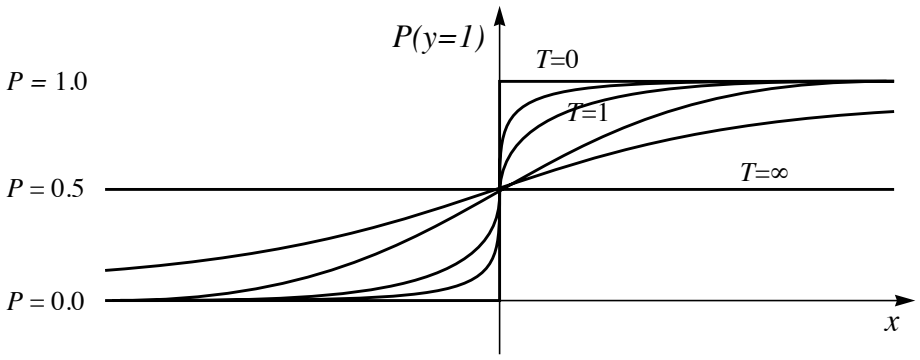


Figure 3.4: Nondeterministic activation functions: Probability of outputing 1 at various temperatures.

network may be able to escape local minima (which can trap deterministic gradient descent procedures like backpropagation), and find global minima instead.

Up to this point we have discussed units whose activation functions have the general form

$$y_j = f(x_j) \qquad \text{where} \qquad x_j = \sum_i y_i w_{ji} \tag{28}$$

This is the most common form of activation function. However, some types of networks — such as Learned Vector Quantization (LVQ) networks, and Radial Basis Function (RBF) networks — include units that are based on another type of activation function, with the general form:

$$y_j = f(x_j) \qquad \text{where} \qquad x_j = \sum_i (y_i - w_{ji})^2 \tag{29}$$

The difference between these two types of units has an intuitive geometric interpretation, illustrated in Figure 3.5. In the first case, x_j is the dot product between an input vector $\boldsymbol{y}$ and a weight vector $\boldsymbol{w}$, so x_j is the length of the projection of $\boldsymbol{y}$ onto $\boldsymbol{w}$, as shown in panel (a). This projection may point either in the same or the opposite direction as $\boldsymbol{w}$, i.e., it may lie either on one side or the other of a hyperplane that is perpendicular to $\boldsymbol{w}$. Inputs that lie on the same side will have $x_j > 0$, while inputs that lie on the opposite side will have $x_j < 0$. Thus, if $y_j = f(x_j)$ is a threshold function, as in Equation (24), then the unit will classify each input in terms of which side of the hyperplane it lies on. (This classification will be fuzzy if a sigmoidal function is used instead of a threshold function.)

By contrast, in the second case, x_j is the Euclidean distance between an input vector $\boldsymbol{y}$ and a weight vector $\boldsymbol{w}$. Thus, the weight represents the center of a spherical distribution in input space, as shown in panel (b). The distance function can be inverted by a function like $y_j = f(x_j) = \exp(-x_j)$, so that an input at the center of the cluster has an activation $y_j = 1$, while an input at an infinite distance has an activation $y_j = 0$.

In either case, such decision regions — defined by hyperplanes or hyperspheres, with either discontinuous or continuous boundaries — can be positioned anywhere in the input space, and used to "carve up" the input space in arbitrary ways. Moreover, a set of such

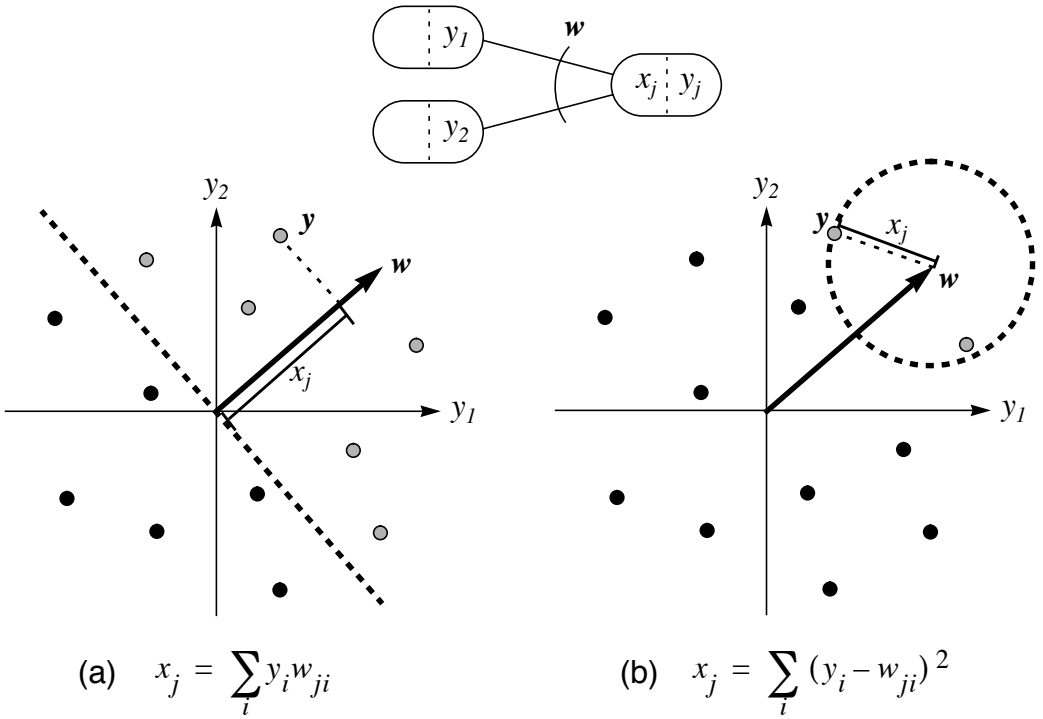


Figure 3.5: Computation of net input. (a) Dot product ⇒ hyperplane; (b) Difference ⇒ hypersphere.

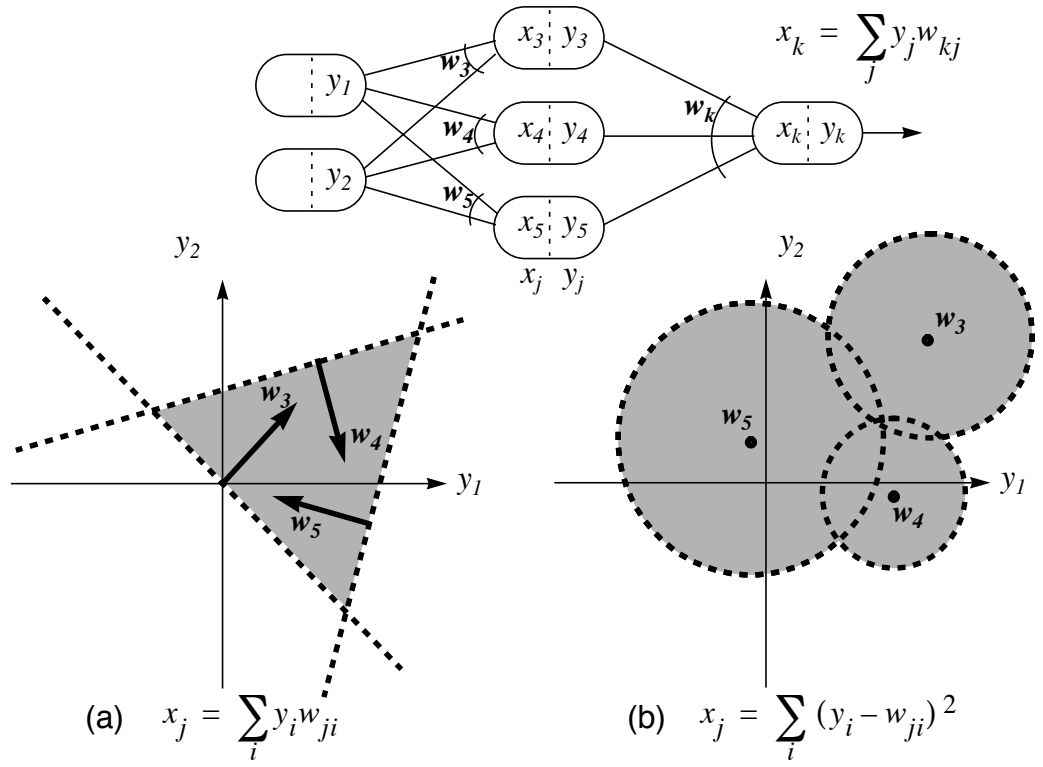


Figure 3.6: Construction of complex functions from (a) hyperplanes, or (b) hyperspheres.

decision regions can be overlapped and combined, to construct any arbitrarily complex function, by including at least one additional layer of threshold (or sigmoidal) units, as illustrated in Figure 3.6. It is the task of a training procedure to adjust the hyperplanes and/or hyperspheres to form a more accurate model of the desired function.

3.2.4. Training

Training a network, in the most general sense, means adapting its connections so that the network exhibits the desired computational behavior for all input patterns. The process usually involves modifying the weights (moving the hyperplanes/hyperspheres); but sometimes it also involves modifying the actual topology of the network, i.e., adding or deleting connections from the network (adding or deleting hyperplanes/hyperspheres). In a sense, weight modification is more general than topology modification, since a network with abundant connections can learn to set any of its weights to zero, which has the same effect as deleting such weights. However, topological changes can improve both generalization and the speed of learning, by constraining the class of functions that the network is capable of learning. Topological changes will be discussed further in Section 3.3.5; in this section we will focus on weight modification.

Finding a set of weights that will enable a given network to compute a given function is usually a nontrivial procedure. An analytical solution exists only in the simplest case of pattern association, i.e., when the network is linear and the goal is to map a set of orthogonal input vectors to output vectors. In this case, the weights are given by

$$w_{ji} = \sum_p \frac{y_i^p t_j^p}{\left\| y^p \right\|^2} \tag{30}$$

where $\boldsymbol{y}$ is the input vector, $\boldsymbol{t}$ is the target vector, and p is the pattern index.

In general, networks are nonlinear and multilayered, and their weights can be trained only by an iterative procedure, such as *gradient descent* on a global performance measure (Hinton 1989). This requires multiple passes of training on the entire training set (rather like a person learning a new skill); each pass is called an *iteration* or an *epoch*. Moreover, since the accumulated knowledge is distributed over all of the weights, the weights must be modified very gently so as not to destroy all the previous learning. A small constant called the *learning rate* (ε) is thus used to control the magnitude of weight modifications. Finding a good value for the learning rate is very important — if the value is too small, learning takes forever; but if the value is too large, learning disrupts all the previous knowledge. Unfortunately, there is no analytical method for finding the optimal learning rate; it is usually optimized empirically, by just trying different values.

Most training procedures, including Equation (30), are essentially variations of the *Hebb Rule* (Hebb 1949), which reinforces the connection between two units if their output activations are correlated:

$$\Delta w_{ji} = \varepsilon y_i y_j \tag{31}$$

By reinforcing the correlation between active pairs of units during training, the network is prepared to activate the second unit if only the first one is known during testing.

One important variation of the above rule is the *Delta Rule* (or the *Widrow-Hoff Rule*), which applies when there is a target value for one of the two units. This rule reinforces the connection between two units if there is a correlation between the first unit's activation y_i and the second unit's error (or potential for error reduction) relative to its target t_j:

$$\Delta w_{ji} = \varepsilon y_i (t_j - y_j) \tag{32}$$

This rule decreases the relative error if y_i contributed to it, so that the network is prepared to compute an output y_j closer to t_j if only the first unit's activation y_i is known during testing. In the context of binary threshold units with a single layer of weights, the Delta Rule is known as the *Perceptron Learning Rule*, and it is guaranteed to find a set of weights representing a perfect solution, if such a solution exists (Rosenblatt 1962). In the context of multilayered networks, the Delta Rule is the basis for the *backpropagation* training procedure, which will be discussed in greater detail in Section 3.4.

Yet another variation of the Hebb Rule applies to the case of spherical functions, as in LVQ and RBF networks:

$$\Delta w_{ji} = \varepsilon (y_i - w_{ji}) y_j \tag{33}$$

This rule moves the spherical center w_{ji} closer to the input pattern y_i if the output class y_j is active.

3.3. A Taxonomy of Neural Networks

Now that we have presented the basic elements of neural networks, we will give an overview of some different types of networks. This overview will be organized in terms of the learning procedures used by the networks. There are three main classes of learning procedures:

- **supervised learning**, in which a "teacher" provides output targets for each input pattern, and corrects the network's errors explicitly;
- **semi-supervised** (or **reinforcement**) **learning**, in which a teacher merely indicates whether the network's response to a training pattern is "good" or "bad"; and
- **unsupervised learning**, in which there is no teacher, and the network must find regularities in the training data by itself.

Most networks fall squarely into one of these categories, but there are also various anomalous networks, such as **hybrid networks** which straddle these categories, and **dynamic networks** whose architectures can grow or shrink over time.

3.3.1. Supervised Learning

Supervised learning means that a "teacher" provides output targets for each input pattern, and corrects the network's errors explicitly. This paradigm can be applied to many types of networks, both feedforward and recurrent in nature. We will discuss these two cases separately.

3.3.1.1. Feedforward Networks

Perceptrons (Rosenblatt 1962) are the simplest type of feedforward networks that use supervised learning. A perceptron is comprised of binary threshold units arranged into layers, as shown in Figure 3.7. It is trained by the Delta Rule given in Equation (32), or variations thereof.

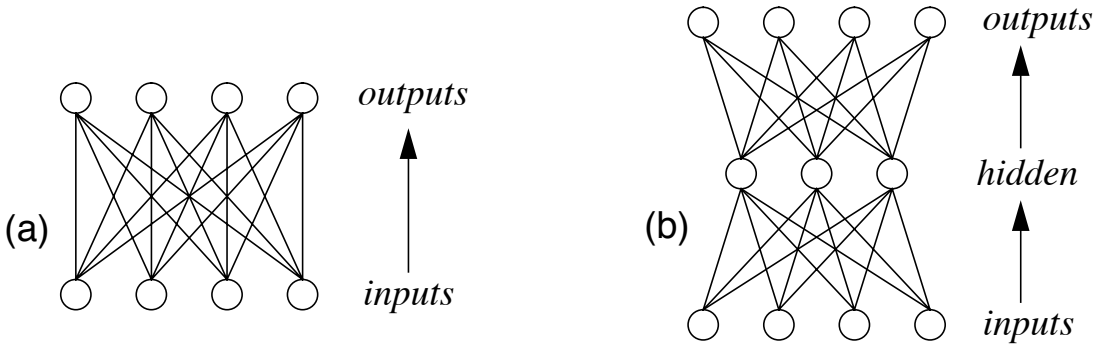


Figure 3.7: Perceptrons. (a) Single layer perceptron; (b) multi-layer perceptron.

In the case of a *single layer perceptron*, as shown in Figure 3.7(a), the Delta Rule can be applied directly. Because a perceptron's activations are binary, this general learning rule reduces to the *Perceptron Learning Rule*, which says that if an input is active ($y_i = 1$) and the output y_j is wrong, then w_{ji} should be either increased or decreased by a small amount ε, depending if the desired output is 1 or 0, respectively. This procedure is guaranteed to find a set of weights to correctly classify the patterns in any training set if the patterns are *linearly separable*, i.e., if they can be separated into two classes by a straight line, as illustrated in Figure 3.5(a). Most training sets, however, are not linearly separable (consider the simple XOR function, for example); in these cases we require multiple layers.

Multi-layer perceptrons (MLPs), as shown in Figure 3.7(b), can theoretically learn any function, but they are more complex to train. The Delta Rule cannot be applied directly to MLPs because there are no targets in the hidden layer(s). However, if an MLP uses continuous rather than discrete activation functions (i.e., sigmoids rather than threshold functions), then it becomes possible to use partial derivatives and the chain rule to derive the influence of any weight on any output activation, which in turn indicates how to modify that weight in order to reduce the network's error. This generalization of the Delta Rule is known as *back-propagation*; it will be discussed further in Section 3.4.

MLPs may have any number of hidden layers, although a single hidden layer is sufficient for many applications, and additional hidden layers tend to make training slower, as the terrain in weight space becomes more complicated. MLPs can also be architecturally constrained in various ways, for instance by limiting their connectivity to geometrically local areas, or by limiting the values of the weights, or tying different weights together.

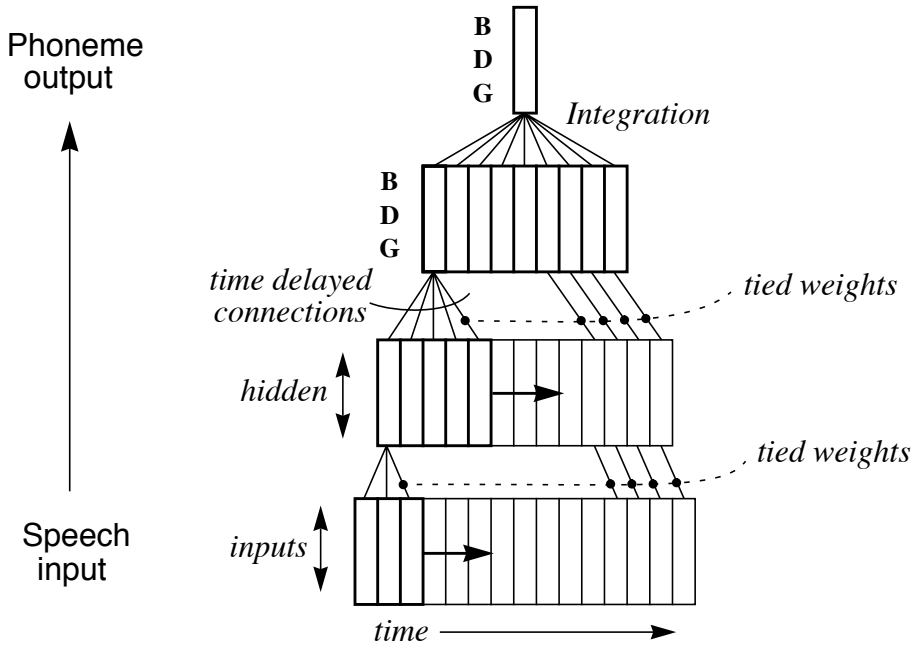


Figure 3.8: Time Delay Neural Network.

One type of constrained MLP which is especially relevant to this thesis is the *Time Delay Neural Network* (TDNN), shown in Figure 3.8. This architecture was initially developed for phoneme recognition (Lang 1989, Waibel et al 1989), but it has also been applied to handwriting recognition (Idan et al, 1992, Bodenhausen and Manke 1993), lipreading (Bregler et al, 1993), and other tasks. The TDNN operates on two-dimensional input fields, where the horizontal dimension is *time*[1]. Connections are "time delayed" to the extent that their connected units are temporally nonaligned. The TDNN has three special architectural features:

1. Its time delays are hierarchically structured, so that higher level units can integrate more temporal context and perform higher level feature detection.
2. Weights are *tied* along the time axis, i.e., corresponding weights at different temporal positions share the same value, so the network has relatively few free parameters, and it can generalize well.
3. The output units temporally integrate the results of local feature detectors distributed over time, so the network is shift invariant, i.e., it can recognize patterns no matter where they occur in time.

1. Assuming the task is speech recognition, or some other task in the temporal domain.

The TDNN is trained using standard backpropagation. The only unusual aspect of training a TDNN is that the tied weights are modified according to their averaged error signal, rather than independently.

Another network that can classify input patterns is a *Learned Vector Quantization* (LVQ) network (Kohonen 1989). An LVQ network is a single-layered network in which the outputs represent classes, and their weights from the inputs represent the centers of hyperspheres, as shown in Figure 3.5(b). Training involves moving the hyperspheres to cover the classes more accurately. Specifically[1], for each training pattern $\boldsymbol{x}$, if the best output y_1 is incorrect, while the second best output y_2 is correct, and if $\boldsymbol{x}$ is near the midpoint between the hyperspheres $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$, then we move $\boldsymbol{w_1}$ toward $\boldsymbol{x}$, and $\boldsymbol{w_2}$ away from $\boldsymbol{x}$:

$$\begin{aligned} \Delta \boldsymbol{w_1} &= +\varepsilon\,(\boldsymbol{x} - \boldsymbol{w_1}) \\ \Delta \boldsymbol{w_2} &= -\varepsilon\,(\boldsymbol{x} - \boldsymbol{w_2}) \end{aligned} \tag{34}$$

3.3.1.2. Recurrent Networks

Hopfield (1982) studied neural networks that implement a kind of content-addressable associative memory. He worked with unstructured networks of binary threshold units with symmetric connections ($w_{ji} = w_{ij}$), in which activations are updated asynchronously; this type of recurrent network is now called a *Hopfield network*. Hopfield showed that if the weights in such a network were modified according to the Hebb Rule, then the training patterns would become *attractors* in state space. In other words, if the network were later presented with a corrupted version of one of the patterns, and the network's activations were updated in a random, asynchronous manner (using the previously trained weights), then the network would gradually reconstruct the whole activation pattern of the closest pattern in state space, and stabilize on that pattern. Hopfield's key insight was to analyze the network's dynamics in terms of a global *energy function*:

$$E = -\frac{1}{2}\sum_i \sum_{j \neq i} w_{ji} y_i y_j \tag{35}$$

which necessarily decreases (or remains the same) when any unit's activation is updated, and which reaches a minimum value for activation patterns corresponding to the stored memories. This implies that the network always settles to a stable state (although it may reach a local minimum corresponding to a spurious memory arising from interference between the stored memories).

A *Boltzmann Machine* (Ackley et al 1985) is a Hopfield network with hidden units, stochastic activations, and simulated annealing in its learning procedure. Each of these features contributes to its exceptional power. The hidden units allow a Boltzmann Machine to find higher order correlations in the data than a Hopfield network can find, so it can learn arbitrarily complex patterns. The stochastic (temperature-based) activations, as shown in Figure 3.4, allow a Boltzmann Machine to escape local minima during state evolution. Simulated annealing (i.e., the use of steadily decreasing temperatures during training) helps the

1. The training algorithm described here is known as LVQ2, an improvement over the original LVQ training algorithm.

network learn more efficiently than if it always used a low temperature, by vigorously "shaking" the network into viable neighborhoods of weight space during early training, and more gently jiggling the network into globally optimal positions during later training. Training a Boltzmann Machine involves modifying the weights so as to reduce the difference between two observed probability distributions:

$$\Delta w_{ji} = \frac{\varepsilon}{T}\left(p_{ij}^{+} - p_{ij}^{-}\right) \tag{36}$$

where T is the temperature, p_{ij}^{+} is the probability (averaged over all environmental inputs and measured at equilibrium) that the ith and jth units are both active when all of the visible units (inputs and/or outputs) have clamped values, and p_{ij}^{-} is the corresponding probability when the system is "free running", i.e., when nothing is clamped. Learning tends to be extremely slow in Boltzmann Machines, not only because it uses gradient descent, but also because at each temperature in the annealing schedule we must wait for the network to come to equilibrium, and then collect lots of statistics about its clamped and unclamped behavior. Nevertheless, Boltzmann Machines are theoretically very powerful, and they have been successfully applied to many problems.

Other types of recurrent networks have a layered structure with connections that feed back to earlier layers. Figure 3.9 shows two examples, known as the *Jordan network* (Jordan 1986) and the *Elman network* (Elman 1990). These networks feature a set of *context* units, whose activations are copied from either the outputs or the hidden units, respectively, and which are then fed forward into the hidden layer, supplementing the inputs. The context units give the networks a kind of decaying memory, which has proven sufficient for learning temporal structure over short distances, but not generally over long distances (Servan-Schreiber et al 1991). These networks can be trained with standard backpropagation, since all of the trainable weights are feedforward weights.

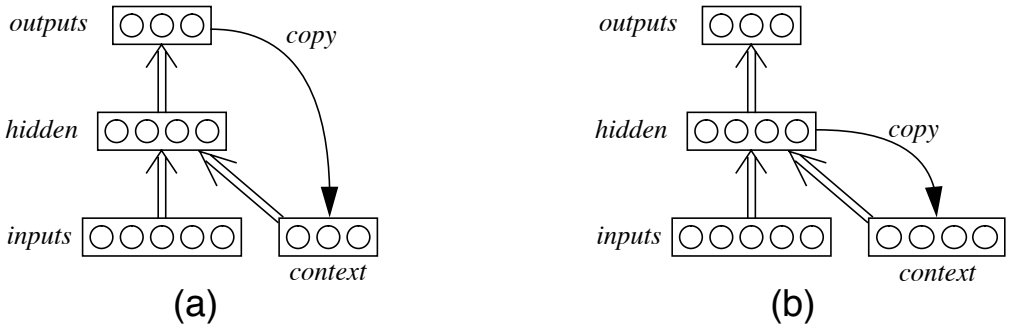


Figure 3.9: Layered recurrent networks. (a) Jordan network; (b) Elman network.

3.3.2. Semi-Supervised Learning

In semi-supervised learning (also called *reinforcement learning*), an external teacher does not provide explicit targets for the network's outputs, but only evaluates the network's behavior as "good" or "bad". Different types of semi-supervised networks are distinguished

not so much by their topologies (which are fairly arbitrary), but by the nature of their environment and their learning procedures. The environment may be either static or dynamic, i.e., the definition of "good" behavior may be fixed or it may change over time; likewise, evaluations may either be deterministic or probabilistic.

In the case of static environments (with either deterministic or stochastic evaluations), networks can be trained by the *associative reward-penalty algorithm* (Barto and Anandan 1985). This algorithm assumes stochastic output units (as in Figure 3.4) which enable the network to try out various behaviors. The problem of semi-supervised learning is reduced to the problem of supervised learning, by setting the training targets to be either the actual outputs or their negations, depending on whether the network's behavior was judged "good" or "bad"; the network is then trained using the Delta Rule, where the targets are compared against the network's mean outputs, and error is backpropagated through the network if necessary.

Another approach, which can be applied to either static or dynamic environments, is to introduce an auxiliary network which tries to model the environment (Munro 1987). This auxiliary network maps environmental data (consisting of both the input and output of the first network) to a reinforcement signal. Thus, the problem of semi-supervised learning is reduced to two stages of supervised learning with known targets — first the auxiliary network is trained to properly model the environment, and then backpropagation can be applied through both networks, so that each output of the original network has a distinct error signal coming from the auxiliary network.

A similar approach, which applies only to dynamic environments, is to enhance the auxiliary network so that it becomes a *critic* (Sutton 1984), which maps environmental data plus the reinforcement signal to a prediction of the future reinforcement signal. By comparing the expected and actual reinforcement signal, we can determine whether the original network's performance exceeds or falls short of expectation, and we can then reward or punish it accordingly.

3.3.3. Unsupervised Learning

In unsupervised learning, there is no teacher, and a network must detect regularities in the input data by itself. Such *self-organizing* networks can be used for compressing, clustering, quantizing, classifying, or mapping input data.

One way to perform unsupervised training is to recast it into the paradigm of supervised training, by designating an artificial target for each input pattern, and applying backpropagation. In particular, we can train a network to reconstruct the input patterns on the output layer, while passing the data through a bottleneck of hidden units. Such a network learns to preserve as much information as possible in the hidden layer; hence the hidden layer becomes a compressed representation of the input data. This type of network is often called an *encoder*, especially when the inputs/outputs are binary vectors. We also say that this network performs *dimensionality reduction*.

Other types of unsupervised networks (usually without hidden units) are trained with Hebbian learning, as in Equation (31). Hebbian learning can be used, for example, to train a sin-

gle linear unit to recognize the familiarity of an input pattern, or by extension to train a set of M linear output units to project an input pattern onto the M principal components of the distribution, thus forming a compressed representation of the inputs on the output layer. With linear units, however, the standard Hebb Rule would cause the weights to grow without bounds, hence this rule must be modified to prevent the weights from growing too large. One of several viable modifications is Sanger's Rule (Sanger 1989):

$$\Delta w_{ji} = \varepsilon \cdot y_j \cdot \left(y_i - \sum_{k=1}^{j} y_k w_{ki} \right) \tag{37}$$

This can be viewed as a form of weight decay (Krogh and Hertz, 1992). This rule uses nonlocal information, but it has the nice property that the M weight vectors $\boldsymbol{w}_j$ converge to the first M principal component directions, in order, normalized to unit length.

Linsker (1986) showed that a modified Hebbian learning rule, when applied to a multilayered network in which each layer is planar and has geometrically local connections to the next layer (as in the human visual system), can automatically develop useful feature detectors, such as center-surround cells and orientation-selective cells, very similar to those found in the human visual system.

Still other unsupervised networks are based on *competitive learning*, in which one output unit is considered the "winner"; these are known as *winner-take-all* networks. The winning unit may be found by lateral inhibitory connections on the output units (which drive down the losing activations to zero), or simply by comparative inspection of the output activations. Competitive learning is useful for clustering the data, in order to classify or quantize input patterns. Note that if the weights $\boldsymbol{w}_i$ to each output unit i are normalized, such that $\|\boldsymbol{w}_i\| = 1$ for all i, then maximizing the net input $\boldsymbol{w}_i \cdot \boldsymbol{y}$ is equivalent to minimizing the difference $\|\boldsymbol{w}_i - \boldsymbol{y}\|$; hence the goal of training can be seen as moving the weight vectors to the centers of hyperspherical input clusters, so as to minimize this distance. The standard competitive learning rule is thus the one given in Equation (33); when outputs are truly winner-take-all, this learning rule simplifies to

$$\Delta w_{j'i} = \varepsilon \cdot (y_i - w_{j'i}) \tag{38}$$

which is applied only to the winning output j'. Unfortunately, with this learning procedure, some units may be so far away from the inputs that they never win, and therefore never learn. Such *dead units* can be avoided by initializing the weights to match actual input samples, or else by relaxing the winner-take-all constraint so that losers learn as well as winners, or by using any of a number of other mechanisms (Hertz, Krogh, & Palmer 1991).

Carpenter and Grossberg (1988) developed networks called ART1 and ART2 (*Adaptive Resonance Theory* networks for binary and continuous inputs, respectively), which support competitive learning in such a way that a new cluster is formed whenever an input pattern is sufficiently different from any existing cluster, according to a *vigilance* parameter. Clusters are represented by individual output units, as usual; but in an ART network the output units are reserved until they are needed. Their network uses a search procedure, which can be implemented in hardware.

Kohonen (1989) developed a competitive learning algorithm which performs *feature mapping*, i.e., mapping patterns from an input space to an output space while preserving topological relations. The learning rule is

$$\Delta w_{ji} = \varepsilon \cdot \Lambda(j, j') \cdot (y_i - w_{ji}) \tag{39}$$

which augments the standard competitive learning rule by a *neighborhood function* $\Lambda(j, j')$, measuring the topological proximity between unit j and the winning unit j', so that units near j' are strongly affected, while distant units are less affected. This can be used, for example, to map two input coefficients onto a 2-dimensional set of output units, or to map a 2-dimensional set of inputs to a different 2-dimensional representation, as occurs in different layers of visual or somatic processing in the brain.

3.3.4. Hybrid Networks

Some networks combine supervised and unsupervised training in different layers. Most commonly, unsupervised training is applied at the lowest layer in order to cluster the data, and then backpropagation is applied at the higher layer(s) to associate these clusters with the desired output patterns. For example, in a *Radial Basis Function* network (Moody and Darken 1989), the hidden layer contains units that describe hyperspheres (trained with a standard competitive learning algorithm), while the output layer computes normalized linear combinations of these receptive field functions (trained with the Delta Rule). The attraction of such hybrid networks is that they reduce the multilayer backpropagation algorithm to the single-layer Delta Rule, considerably reducing training time. On the other hand, since such networks are trained in terms of independent modules rather than as an integrated whole, they have somewhat less accuracy than networks trained entirely with backpropagation.

3.3.5. Dynamic Networks

All of the networks discussed so far have a static architecture. But there are also *dynamic networks*, whose architecture can change over time, in order to attain optimal performance. Changing an architecture involves either deleting or adding elements (weights and/or units) in the network; these opposite approaches are called *pruning* and *construction*, respectively. Of these two approaches, pruning tends to be simpler, as it involves merely ignoring selected elements; but constructive algorithms tend to be faster, since the networks are small for much of their lives.

Pruning of course requires a way to identify the least useful elements in the network. One straightforward technique is to delete the weights with the smallest magnitude; this can improve generalization, but sometimes it also eliminates the wrong weights (Hassibi and Stork 1993). A more complex but more reliable approach, called *Optimal Brain Damage* (Le Cun et al, 1990b), identifies the weights whose removal will cause the least increase in the network's output error function; this requires the calculation of second-derivative information.

Among constructive algorithms, the *Cascade Correlation* algorithm (Fahlman and Lebiere 1990) is one of the most popular and effective. This algorithm starts with no hidden units, but gradually adds them (in depth-first fashion) as long as they help to cut down any remaining output error. At each stage of training, all previously trained weights in the network are frozen, and a pool of new candidate units are connected to all existing non-output units; each candidate unit is trained to maximize the correlation between the unit's output and the network's residual error, and then the most effective unit is fully integrated into the network (while the other candidates are discarded), and its weights to the output layer are fine-tuned. This process is repeated until the network has acceptable performance. The Cascade Correlation algorithm can quickly construct compact, powerful networks that exhibit excellent performance.

Bodenhausen (1994) has developed a constructive algorithm called *Automatic Structure Optimization*, designed for spacio-temporal tasks such as speech recognition and online handwriting recognition, especially given limited training data. The ASO algorithm starts with a small network, and adds more resources (including connections, time delays, hidden units, and state units) in a class-dependent way, under the guidance of confusion matrices obtained by cross-validation during training, in order to minimize the overall classification error. The ASO algorithm automatically optimized the architecture of MS-TDNNs, achieving results that were competitive with state-of-the-art systems that had been optimized by hand.

3.4. Backpropagation

Backpropagation, also known as *Error Backpropagation* or the *Generalized Delta Rule*, is the most widely used supervised training algorithm for neural networks. Because of its importance, we will discuss it in some detail in this section. We begin with a full derivation of the learning rule.

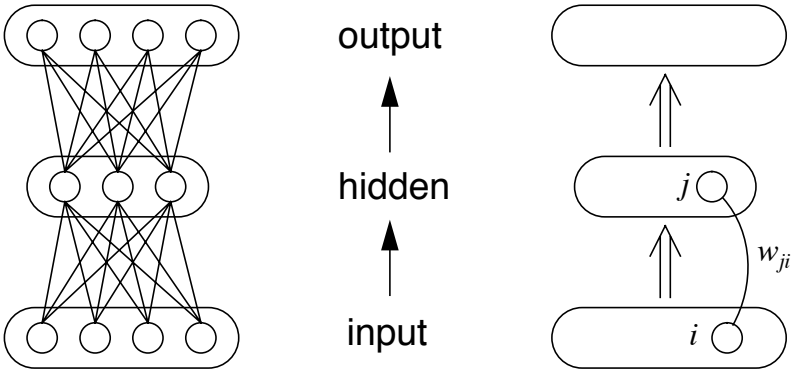


Figure 3.10: A feedforward neural network, highlighting the connection from unit i to unit j.

Suppose we have a multilayered feedforward network of nonlinear (typically sigmoidal) units, as shown in Figure 3.10. We want to find values for the weights that will enable the network to compute a desired function from input vectors to output vectors. Because the units compute nonlinear functions, we cannot solve for the weights analytically; so we will instead use a gradient descent procedure on some global error function E.

Let us define i, j, and k as arbitrary unit indices, O as the set of output units, p as training pattern indices (where each training pattern contains an input vector and output target vector), x_j^p as the net input to unit j for pattern p, y_j^p as the output activation of unit j for pattern p, w_{ji} as the weight from unit i to unit j, t_j^p as the target activation for unit j in pattern p (for $j \in O$), E^p as the global output error for training pattern p, and E as the global error for the entire training set. Assuming the most common type of network, we have

$$x_j^p = \sum_i w_{ji} y_i^p \tag{40}$$

$$y_j^p = \sigma(x_j^p) = \frac{1}{1 + e^{-x_j^p}} \tag{41}$$

It is essential that this activation function $y_j^p = \sigma(x_j^p)$ be differentiable, as opposed to non-differentiable as in a simple threshold function, because we will be computing its gradient in a moment.

The choice of error function is somewhat arbitrary[1]; let us assume the Sum Squared Error function

$$E^p = \frac{1}{2} \sum_j (y_j^p - t_j^p)^2 \qquad \text{where} \qquad j \in O \tag{42}$$

and

$$E = \sum_p E^p \tag{43}$$

We want to modify each weight w_{ji} in proportion to its influence on the error E, in the direction that will reduce E:

$$\Delta^p w_{ji} = -\varepsilon \cdot \frac{\partial E^p}{\partial w_{ji}} \tag{44}$$

where ε is a small constant, called the *learning rate*.

1. Popular choices for the global error function include Sum Squared Error: $E = \frac{1}{2} \sum_j (y_j - t_j)^2$; Cross Entropy: $E = -\sum_j (t_j \log y_j) + (1 - t_j) \log (1 - y_j)$; McClelland Error: $E = -\sum_j \log (1 - (t_j - y_j)^2)$; and the Classification Figure of Merit: $E = f(d)$ where $d = y_{\bar{c}} - y_c$ = the difference between the best incorrect output and the correct output, for example $E = (d + 1)^2$.

By the Chain Rule, and from Equations (41) and (40), we can expand this as follows:

$$\frac{\partial E^p}{\partial w_{ji}} = \frac{\partial E^p}{\partial y_j^p} \cdot \frac{\partial y_j^p}{\partial x_j^p} \cdot \frac{\partial x_j^p}{\partial w_{ji}} \qquad (45)$$

$$\overset{def.}{\downarrow} \quad \downarrow \quad \downarrow$$

$$= \gamma_j^p \cdot \sigma'(x_j^p) \cdot y_i^p$$

The first of these three terms, which introduces the shorthand definition $\gamma_j^p = \partial E^p / \partial y_j^p$, remains to be expanded. Exactly how it is expanded depends on whether j is an output unit or not. If j is an output unit, then from Equation (42) we have

$$j \in O \quad \Rightarrow \quad \gamma_j^p = \frac{\partial E^p}{\partial y_j^p} = y_j^p - t_j^p \qquad (46)$$

But if j is not an output unit, then it directly affects a set of units $k \in \text{out}(j)$, as illustrated in Figure 3.11, and by the Chain Rule we obtain

$$j \notin O \quad \Rightarrow \quad \gamma_j^p = \frac{\partial E^p}{\partial y_j^p} = \sum_{k \in \text{out}(j)} \frac{\partial E^p}{\partial y_k^p} \cdot \frac{\partial y_k^p}{\partial x_k^p} \cdot \frac{\partial x_k^p}{\partial y_j^p} \qquad (47)$$

$$\downarrow \quad \downarrow \quad \downarrow$$

$$= \sum_{k \in \text{out}(j)} \gamma_k^p \cdot \sigma'(x_k^p) \cdot w_{kj}$$

The recursion in this equation, in which γ_j^p refers to γ_k^p, says that the γ's (and hence Δw's) in each layer can be derived directly from the γ's in the next layer. Thus, we can derive all the γ's in a multilayer network by starting at the output layer (using Equation 46) and working our way backwards towards the input layer, one layer at a time (using Equation 47). This learning procedure is called "backpropagation" because the error terms (γ's) are propagated through the network in this backwards direction.

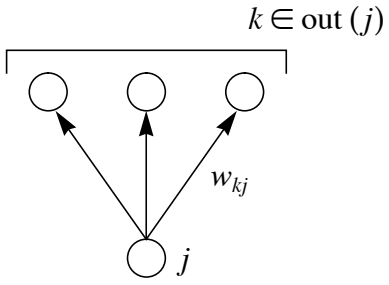


Figure 3.11: If unit *j* is not an output unit, then it directly affects some units *k* in the next layer.

To summarize the learning rule, we have

$$\Delta^p w_{ji} = -\varepsilon \cdot \gamma_j^p \cdot \sigma'(x_j^p) \cdot y_i^p \tag{48}$$

where

$$\gamma_j^p = \frac{\partial E^p}{\partial y_j^p} = \begin{cases} (y_j^p - t_j^p) & \text{if} \quad j \in O \\ \sum_{k \in \text{out}(j)} \gamma_k^p \cdot \sigma'(x_k^p) \cdot w_{kj} & \text{if} \quad j \notin O \end{cases} \tag{49}$$

Or, equivalently, if we wish to define

$$\delta_j^p = \frac{\partial E^p}{\partial x_j^p} = \gamma_j^p \cdot \sigma'(x_j^p) \tag{50}$$

then we have

$$\Delta^p w_{ji} = -\varepsilon \cdot \delta_j^p \cdot y_i^p \tag{51}$$

where

$$\delta_j^p = \frac{\partial E^p}{\partial x_j^p} = \begin{cases} (y_j^p - t_j^p) \cdot \sigma'(x_j^p) & \text{if} \quad j \in O \\ \sum_{k \in \text{out}(j)} (\delta_k^p \cdot w_{kj}) \cdot \sigma'(x_j^p) & \text{if} \quad j \notin O \end{cases} \tag{52}$$

Backpropagation is a much faster learning procedure than the Boltzmann Machine training algorithm, but it can still take a long time for it to converge to an optimal set of weights. Learning may be accelerated by increasing the learning rate ε, but only up to a certain point, because when the learning rate becomes too large, weights become excessive, units become saturated, and learning becomes impossible. Thus, a number of other heuristics have been developed to accelerate learning. These techniques are generally motivated by an intuitive image of backpropagation as a gradient descent procedure. That is, if we envision a hilly landscape representing the error function E over weight space, then backpropagation tries to find a local minimum value of E by taking incremental steps Δw_{ji} down the current hillside, i.e., in the direction $-\partial E^p / \partial w_{ji}$. This image helps us see, for example, that if we take too large of a step, we run the risk of moving so far down the current hillside that we find ourselves shooting up some other nearby hillside, with possibly a higher error than before.

Bearing this image in mind, one common heuristic for accelerating the learning process is known as *momentum* (Rumelhart et al 1986), which tends to push the weights further along in the most recently useful direction:

$$\Delta w_{ji}(t) = \left(-\varepsilon \cdot \frac{\partial E^p}{\partial w_{ji}}\right) + \left(\alpha \cdot \Delta w_{ji}(t-1)\right) \tag{53}$$

where α is the momentum constant, usually between 0.50 and 0.95. This heuristic causes the step size to steadily increase as long as we keep moving down a long gentle valley, and also to recover from this behavior when the error surface forces us to change direction. A more elaborate and more powerful heuristic is to use second-derivative information to estimate how far down the hillside to travel; this is used in techniques such as *conjugate gradient* (Barnard 1992) and *quickprop* (Fahlman 1988).

Ordinarily the weights are updated after each training pattern (this is called *online training*. But sometimes it is more effective to update the weights only after accumulating the gradients over a whole batch of training patterns (this is called *batch training*), because by superimposing the error landscapes for many training patterns, we can find a direction to move which is best for the whole group of patterns, and then confidently take a larger step in that direction. Batch training is especially helpful when the training patterns are uncorrelated (for then it eliminates the waste of Brownian motion), and when used in conjunction with aggressive heuristics like quickprop which require accurate estimates of the landscape's surface.

Because backpropagation is a simple gradient descent procedure, it is unfortunately susceptible to the problem of local minima, i.e., it may converge upon a set of weights that are locally optimal but globally suboptimal. Experience has shown that local minima tend to cause more problems for artificial domains (as in boolean logic) than for real domains (as in perceptual processing), reflecting a difference in terrain in weight space. In any case, it is possible to deal with the problem of local minima by adding noise to the weight modifications.

3.5. Relation to Statistics

Neural networks have a close relationship to many standard statistical techniques. In this section we discuss some of these commonalities.

One of the most important tasks in statistics is the classification of data. Suppose we want to classify an input vector x into one of two classes, c_1 and c_2. Obviously our decision should correspond to the class with the highest probability of being correct, i.e., we should decide on class c_1 if $P(c_1|x) > P(c_2|x)$. Normally these posterior probabilities are not known, but the "inverse" information, namely the probability distributions $P(x|c_1)$ and $P(x|c_2)$, may be known. We can convert between posterior probabilities and distributions using Bayes Rule:

$$P(c_i|x) = \frac{P(x|c_i) \cdot P(c_i)}{P(x)} \qquad \text{where} \qquad P(x) = \sum_i P(x|c_i)\,P(c_i) \tag{54}$$

It follows directly that we should choose class c_1 if

$$P(x|c_1)\,P(c_1) > P(x|c_2)\,P(c_2) \tag{55}$$

This criterion is known as the *Bayes Decision Rule*. Given perfect knowledge of the distributions $P(x|c_i)$ and priors $P(c_i)$, this decision rule is guaranteed to minimize the classification error rate.

Typically, however, the distributions, priors, and posteriors are all unknown, and all we have is a collection of sample data points. In this case, we must analyze and model the data in order to classify new data accurately. If the existing data is labeled, then we can try to estimate either the posterior probabilities $P(c|x)$, or the distributions $P(x|c)$ and priors $P(c)$, so that we can use Bayes Decision Rule; alternatively, we can try to find boundaries that separate the classes, without trying to model their probabilities explicitly. If the data is unlabeled, we can first try to cluster it, in order to identify meaningful classes. Each of the above tasks can be performed either by a statistical procedure or by a neural network.

For example, if we have labeled data, and we wish to perform Bayesian classification, there are many statistical techniques available for modeling the data (Duda and Hart 1973). These include both parametric and nonparametric approaches. In the parametric approach, we assume that a distribution $P(x|c)$ has a given parametric form (e.g., a gaussian density), and then try to estimate its parameters; this is commonly done by a procedure called *Maximum Likelihood* estimation, which finds the parameters that maximize the likelihood of having generated the observed data. In the non-parametric approach, we may use a volumetric technique called *Parzen windows* to estimate the local density of samples at any point x; the robustness of this technique is often improved by scaling the local volume around x so that it always contains k samples, in a variation called *k-nearest neighbor estimation*. (Meanwhile, the priors $P(c)$ can be estimated simply by counting.) Alternatively, the posterior probability $P(c|x)$ can also be estimated using nonparametric techniques, such as the *k-nearest neighbor rule*, which classifies x in agreement with the majority of its k nearest neighbors.

A neural network also supports Bayesian classification by forming a model of the training data. More specifically, when a multilayer perceptron is asymptotically trained as a 1-of-N classifier using the mean squared error (MSE) or similar error function, its output activations learn to approximate the posterior probability $P(c|x)$, with an accuracy that improves with the size of the training set. A proof of this important fact can be found in Appendix B.

Another way to use labeled training data is to find boundaries that separate the classes. In statistics, this can be accomplished by a general technique called *discriminant analysis*. An important instance of this is *Fisher's linear discriminant*, which finds a line that gives the best discrimination between classes when data points are projected onto this line. This line is equivalent to the weight vector of a single layer perceptron with a single output that has been trained to discriminate between the classes, using the Delta Rule. In either case, the classes will be optimally separated by a hyperplane drawn perpendicular to the line or the weight vector, as shown in Figure 3.5(a).

Unlabeled data can be clustered using statistical techniques — such as *nearest-neighbor clustering, minimum squared error clustering,* or *k-means clustering* (Krishnaiah and Kanal

1982) — or alternatively by neural networks that are trained with competitive learning. In fact, k-means clustering is exactly equivalent to the standard competitive learning rule, as given in Equation (38), when using batch updating (Hertz et al 1991).

When analyzing high-dimensional data, it is often desirable to reduce its dimensionality, i.e., to project it into a lower-dimensional space while preserving as much information as possible. Dimensionality reduction can be performed by a statistical technique called *Principal Components Analysis* (PCA), which finds a set of M orthogonal vectors that account for the greatest variance in the data (Jolliffe 1986). Dimensionality reduction can also be performed by many types of neural networks. For example, a single layer perceptron, trained by an unsupervised competitive learning rule called *Sanger's Rule* (Equation 37), yields weights that equal the principal components of the training data, so that the network's outputs form a compressed representation of any input vector. Similarly, an encoder network — i.e., a multilayer perceptron trained by backpropagation to reproduce its input vectors on its output layer — forms a compressed representation of the data in its hidden units.

It is sometimes claimed that neural networks are simply a new formulation of old statistical techniques. While there is considerable overlap between these two fields, neural networks are attractive in their own right because they offer a general, uniform, and intuitive framework which can be applied equally well in statistical and non-statistical contexts.

4. Related Research

4.1. Early Neural Network Approaches

Because speech recognition is basically a pattern recognition problem, and because neural networks are good at pattern recognition, many early researchers naturally tried applying neural networks to speech recognition. The earliest attempts involved highly simplified tasks, e.g., classifying speech segments as voiced/unvoiced, or nasal/fricative/plosive. Success in these experiments encouraged researchers to move on to phoneme classification; this task became a proving ground for neural networks as they quickly achieved world-class results. The same techniques also achieved some success at the level of word recognition, although it became clear that there were scaling problems, which will be discussed later.

There are two basic approaches to speech classification using neural networks: static and dynamic, as illustrated in Figure 4.1. In static classification, the neural network sees all of the input speech at once, and makes a single decision. By contrast, in dynamic classification, the neural network sees only a small window of the speech, and this window slides over the input speech while the network makes a series of local decisions, which have to be integrated into a global decision at a later time. Static classification works well for phoneme recognition, but it scales poorly to the level of words or sentences; dynamic classification scales better. Either approach may make use of recurrent connections, although recurrence is more often found in the dynamic approach.

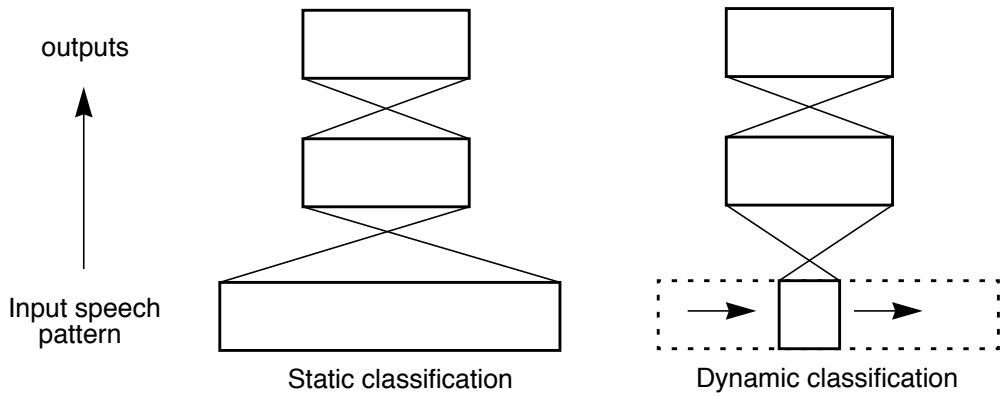


Figure 4.1: Static and dynamic approaches to classification.

In the following sections we will briefly review some representative experiments in phoneme and word classification, using both static and dynamic approaches.

4.1.1. Phoneme Classification

Phoneme classification can be performed with high accuracy by using either static or dynamic approaches. Here we review some typical experiments using each approach.

4.1.1.1. Static Approaches

A simple but elegant experiment was performed by Huang & Lippmann (1988), demonstrating that neural networks can form complex decision surfaces from speech data. They applied a multilayer perceptron with only 2 inputs, 50 hidden units, and 10 outputs, to Peterson & Barney's collection of vowels produced by men, women, & children, using the first two formants of the vowels as the input speech representation. After 50,000 iterations of training, the network produced the decision regions shown in Figure 4.2. These decision regions are nearly optimal, resembling the decision regions that would be drawn by hand, and they yield classification accuracy comparable to that of more conventional algorithms, such as k-nearest neighbor and Gaussian classification.

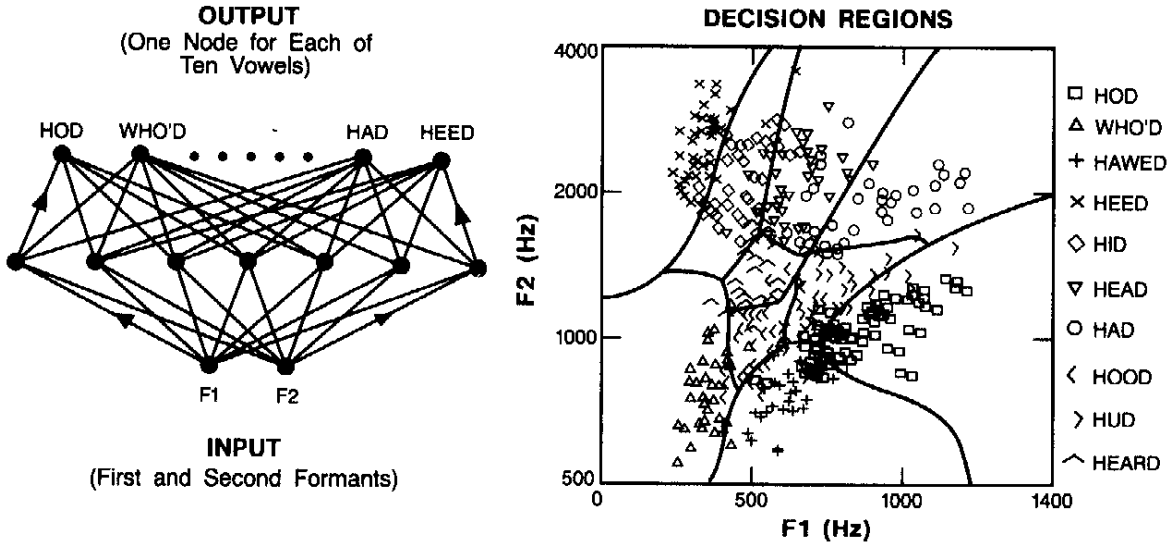


Figure 4.2: Decision regions formed by a 2-layer perceptron using backpropagation training and vowel formant data. (From Huang & Lippmann, 1988.)

In a more complex experiment, Elman and Zipser (1987) trained a network to classify the vowels /a,i,u/ and the consonants /b,d,g/ as they occur in the utterances ba,bi,bu; da,di,du; and ga,gi,gu. Their network input consisted of 16 spectral coefficients over 20 frames (covering an entire 64 msec utterance, centered by hand over the consonant's voicing onset); this was fed into a hidden layer with between 2 and 6 units, leading to 3 outputs for either vowel or consonant classification. This network achieved error rates of roughly 0.5% for vowels and 5.0% for consonants. An analysis of the hidden units showed that they tend to be fea-

ture detectors, discriminating between important classes of sounds, such as consonants versus vowels.

Among the most difficult of classification tasks is the so-called E-set, i.e., discriminating between the rhyming English letters "B, C, D, E, G, P, T, V, Z". Burr (1988) applied a static network to this task, with very good results. His network used an input window of 20 spectral frames, automatically extracted from the whole utterance using energy information. These inputs led directly to 9 outputs representing the E-set letters. The network was trained and tested using 180 tokens from a single speaker. When the early portion of the utterance was oversampled, effectively highlighting the disambiguating features, recognition accuracy was nearly perfect.

4.1.1.2. Dynamic Approaches

In a seminal paper, Waibel et al (1987=1989) demonstrated excellent results for phoneme classification using a Time Delay Neural Network (TDNN), shown in Figure 4.3. This architecture has only 3 and 5 delays in the input and hidden layer, respectively, and the final output is computed by integrating over 9 frames of phoneme activations in the second hidden layer. The TDNN's design is attractive for several reasons: its compact structure economizes on weights and forces the network to develop general feature detectors; its hierarchy of delays optimizes these feature detectors by increasing their scope at each layer; and its temporal integration at the output layer makes the network shift invariant (i.e., insensitive to the exact positioning of the speech). The TDNN was trained and tested on 2000 samples of /b,d,g/ phonemes manually excised from a database of 5260 Japanese words. The TDNN achieved an error rate of 1.5%, compared to 6.5% achieved by a simple HMM-based recognizer.

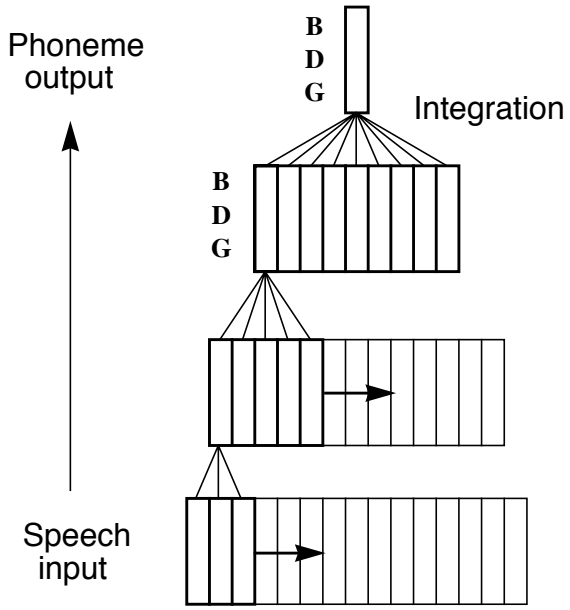


Figure 4.3: Time Delay Neural Network.

In later work (Waibel 1989a), the TDNN was scaled up to recognize all 18 Japanese consonants, using a modular approach which significantly reduced training time while giving slightly better results than a simple TDNN with 18 outputs. The modular approach consisted of training separate TDNNs on small subsets of the phonemes, and then combining these networks into a larger network, supplemented by some "glue" connections which received a little extra training while the primary modules remained fixed. The integrated network achieved an error rate of 4.1% on the 18 phonemes, compared to 7.3% achieved by a relatively advanced HMM-based recognizer.

McDermott & Katagiri (1989) performed an interesting comparison between Waibel's TDNN and Kohonen's LVQ2 algorithm, using the same /b,d,g/ database and similar conditions. The LVQ2 system was trained to quantize a 7-frame window of 16 spectral coefficients into a codebook of 150 entries, and during testing the distance between each input window and the nearest codebook vector was integrated over 9 frames, as in the TDNN, to produce a shift-invariant phoneme hypothesis. The LVQ2 system achieved virtually the same error rate as the TDNN (1.7% vs. 1.5%), but LVQ2 was much faster during training, slower during testing, and more memory-intensive than the TDNN.

In contrast to the feedforward networks described above, recurrent networks are generally trickier to work with and slower to train; but they are also theoretically more powerful, having the ability to represent temporal sequences of unbounded depth, without the need for artificial time delays. Because speech is a temporal phenomenon, many researchers consider recurrent networks to be more appropriate than feedforward networks, and some researchers have actually begun applying recurrent networks to speech.

Prager, Harrison, & Fallside (1986) made an early attempt to apply Boltzmann machines to an 11-vowel recognition task. In a typical experiment, they represented spectral inputs with 2048 binary inputs, and vowel classes with 8 binary outputs; their network also had 40 hidden units, and 7320 weights. After applying simulated annealing for many hours in order to train on 264 tokens from 6 speakers, the Boltzmann machine attained a multi-speaker error rate of 15%. This and later experiments suggested that while Boltzmann machines can give good accuracy, they are impractically slow to train.

Watrous (1988) applied recurrent networks to a set of basic discrimination tasks. In his system, framewise decisions were temporally integrated via recurrent connections on the output units, rather than by explicit time delays as in a TDNN; and his training targets were Gaussian-shaped pulses, rather than constant values, to match the ramping behavior of his recurrent outputs. Watrous obtained good results on a variety of discrimination tasks, after optimizing the non-output delays and sizes of his networks separately for each task. For example, the classification error rate was 0.8% for the consonants /b,d,g/, 0.0% for the vowels /a,i,u/, and 0.8% for the word pair "rapid/rabid".

Robinson and Fallside (1988) applied another kind of recurrent network, first proposed by Jordan (1986), to phoneme classification. In this network, output activations are copied to a "context" layer, which is then fed back like additional inputs to the hidden layer (as shown in Figure 3.9). The network was trained using "back propagation through time", an algorithm first suggested by Rumelhart et al (1986), which unfolds or replicates the network at each moment of time. Their recurrent network outperformed a feedforward network with

comparable delays, achieving 22.7% versus 26.0% error for speaker-dependent recognition, and 30.8% versus 40.8% error for multi-speaker recognition. Training time was reduced to a reasonable level by using a 64-processor array of transputers.

4.1.2. Word Classification

Word classification can also be performed with either static or dynamic approaches, although dynamic approaches are better able to deal with temporal variability over the duration of a word. In this section we review some experiments with each approach.

4.1.2.1. Static Approaches

Peeling and Moore (1987) applied MLPs to digit recognition with excellent results. They used a static input buffer of 60 frames (1.2 seconds) of spectral coefficients, long enough for the longest spoken word; briefer words were padded with zeros and positioned randomly in the 60-frame buffer. Evaluating a variety of MLP topologies, they obtained the best performance with a single hidden layer with 50 units. This network achieved accuracy near that of an advanced HMM system: error rates were 0.25% versus 0.2% in speaker-dependent experiments, or 1.9% versus 0.6% for multi-speaker experiments, using a 40-speaker database of digits from RSRE. In addition, the MLP was typically five times faster than the HMM system.

Kammerer and Kupper (1988) applied a variety of networks to the TI 20-word database, finding that a single-layer perceptron outperformed both multi-layer perceptrons and a DTW template-based recognizer in many cases. They used a static input buffer of 16 frames, into which each word was linearly normalized, with 16 2-bit coefficients per frame; performance improved slightly when the training data was augmented by temporally distorted tokens. Error rates for the SLP versus DTW were 0.4% versus 0.7% in speaker-dependent experiments, or 2.7% versus 2.5% for speaker-independent experiments.

Lippmann (1989) points out that while the above results seem impressive, they are mitigated by evidence that these small-vocabulary tasks are not really very difficult. Burton et al (1985) demonstrated that a simple recognizer based on whole-word vector quantization, without time alignment, can achieve speaker-dependent error rates as low as 0.8% for the TI 20-word database, or 0.3 for digits. Thus it is not surprising that simple networks can achieve good results on these tasks, in which temporal information is not very important.

Burr (1988) applied MLPs to the more difficult task of alphabet recognition. He used a static input buffer of 20 frames, into which each spoken letter was linearly normalized, with 8 spectral coefficients per frame. Training on three sets of the 26 spoken letters and testing on a fourth set, an MLP achieved an error rate of 15% in speaker-dependent experiments, matching the accuracy of a DTW template-based approach.

4.1.2.2. Dynamic Approaches

Lang et al (1990) applied TDNNs to word recognition, with good results. Their vocabulary consisting of the highly confusable spoken letters “B, D, E, V”. In early experiments,

training and testing were simplified by representing each word by a 144 msec segment centered on its vowel segment, where the words differed the most from each other. Using such pre-segmented data, the TDNN achieved a multispeaker error rate of 8.5%. In later experiments, the need for pre-segmentation was avoided by classifying a word according to the output that received the highest activation at any position of the input window relative to the whole utterance; and training used 216 msec segments roughly centered on vowel onsets according to an automatic energy-based segmentation technique. In this mode, the TDNN achieved an error rate of 9.5%. The error rate fell to 7.8% when the network received additional negative training on counter examples randomly selected from the background "E" sounds. This system compared favorably to an HMM which achieved about 11% error on the same task (Bahl et al 1988).

Tank & Hopfield (1987) proposed a "Time Concentration" network, which represents words by a weighted sum of evidence that is delayed, with proportional dispersion, until the end of the word, so that activation is concentrated in the correct word's output at the end of the utterance. This system was inspired by research on the auditory processing of bats, and a working prototype was actually implemented in parallel analog hardware. Unnikrishnan et al (1988) reported good results for this network on simple digit strings, although Gold (1988) obtained results no better than a standard HMM when he applied a hierarchical version of the network to a large speech database.

Among the early studies using recurrent networks, Prager, Harrison, & Fallside (1986) configured a Boltzmann machine to copy the output units into "state" units which were fed back into the hidden layer, as in a so-called Jordan network, thereby representing a kind of first-order Markov model. After several days of training, the network was able to correctly identify each of the words in its two training sentences. Other researchers have likewise obtained good results with Boltzmann machines, but only after an exorbitant amount of training.

Franzini, Witbrock, & Lee (1989) compared the performance of a recurrent network and a feedforward network on a digit recognition task. The feedforward network was an MLP with a 500 msec input window, while the recurrent network had a shorter 70 msec input window but a 500 msec state buffer. They found no significant difference in the recognition accuracy of these systems, suggesting that it's important only that a network have some form of memory, regardless of whether it's represented as a feedforward input buffer or a recurrent state layer.

4.2. The Problem of Temporal Structure

We have seen that phoneme recognition can easily be performed using either static or dynamic approaches. We have also seen that word recognition can likewise be performed with either approach, although dynamic approaches now become preferable because the wider temporal variability in a word implies that invariances are localized, and that local features should be temporally integrated. Temporal integration itself can easily be performed by a network (e.g., in the output layer of a TDNN), as long as the operation can be

described statically (to match the network's fixed resources); but as we consider larger chunks of speech, with greater temporal variability, it becomes harder to map that variability into a static framework. As we continue scaling up the task from word recognition to sentence recognition, temporal variability not only becomes more severe, but it also acquires a whole new dimension — that of compositional structure, as governed by a grammar.

The ability to compose structures from simpler elements — implying the usage of some sort of variables, binding, modularity, and rules — is clearly required in any system that claims to support natural language processing (Pinker and Prince 1988), not to mention general cognition (Fodor and Pylyshyn 1988). Unfortunately, it has proven very difficult to model compositionality within the pure connectionist framework, although a number of researchers have achieved some early, limited success along these lines. Touretzky and Hinton (1988) designed a distributed connectionist production system, which dynamically retrieves elements from working memory and uses their components to contruct new states. Smolensky (1990) proposed a mechanism for performing variable binding, based on tensor products. Servan-Schreiber, Cleeremans, and McClelland (1991) found that an Elman network was capable of learning some aspects of grammatical structure. And Jain (1992) designed a modular, highly structured connectionist natural language parser that compared favorably to a standard LR parser.

But each of these systems is exploratory in nature, and their techniques are not yet generally applicable. It is clear that connectionist research in temporal and compositional modeling is still in its infancy, and it is premature to rely on neural networks for temporal modeling in a speech recognition system.

4.3. NN-HMM Hybrids

We have seen that neural networks are excellent at acoustic modeling and parallel implementations, but weak at temporal and compositional modeling. We have also seen that Hidden Markov Models are good models overall, but they have some weaknesses too. In this section we will review ways in which researchers have tried to combine these two approaches into various hybrid systems, capitalizing on the strengths of each approach. Much of the research in this section was conducted at the same time that this thesis was being written.

4.3.1. NN Implementations of HMMs

Perhaps the simplest way to integrate neural networks and Hidden Markov Models is to simply implement various pieces of HMM systems using neural networks. Although this does not improve the accuracy of an HMM, it does permit it to be parallelized in a natural way, and incidentally showcases the flexibility of neural networks.

Lippmann and Gold (1987) introduced the *Viterbi Net*, illustrated in Figure 4.4, which is a neural network that implements the Viterbi algorithm. The input is a temporal sequence of speech frames, presented one at a time, and the final output (after T time frames) is the

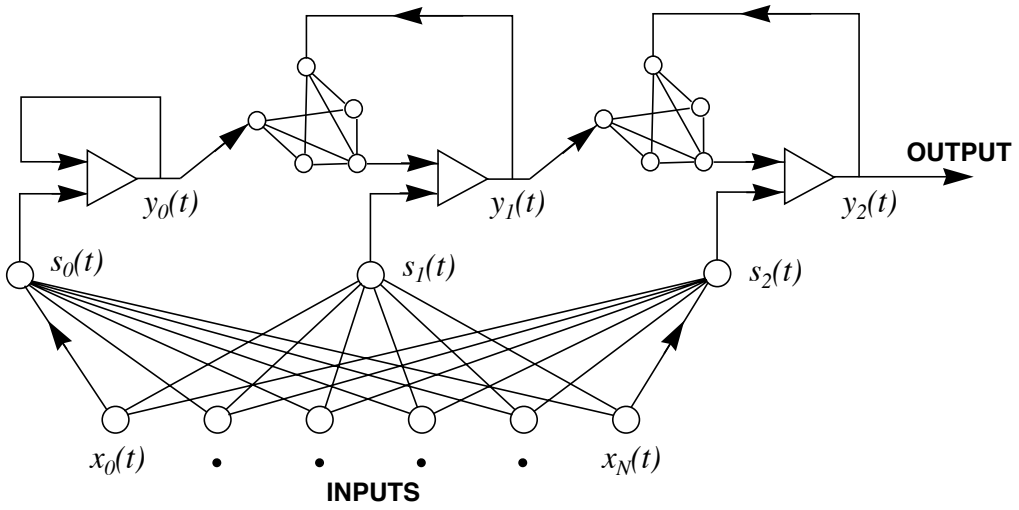


Figure 4.4: Viterbi Net: a neural network that implements the Viterbi algorithm.

cumulative score along the Viterbi alignment path, permitting isolated word recognition via subsequent comparison of the outputs of several Viterbi Nets running in parallel. (The Viterbi Net cannot be used for continuous speech recognition, however, because it yields no backtrace information from which the alignment path could be recovered.) The weights in the lower part of the Viterbi Net are preassigned in such a way that each node s_i computes the local score for state i in the current time frame, implementing a Gaussian classifier. The knotlike upper networks compute the maximum of their two inputs. The triangular nodes are threshold logic units that simply sum their two inputs (or output zero if the sum is negative), and delay the output by one time frame, for synchronization purposes. Thus, the whole network implements a left-to-right HMM with self-transitions, and the final output $y_F(T)$ represents the cumulative score in state F at time T along the optimal alignment path. It was tested on 4000 word tokens from the 9-speaker 35-word Lincoln Stress-Style speech database, and obtained results essentially identical with a standard HMM (0.56% error).

In a similar spirit, Bridle (1990) introduced the *AlphaNet*, which is a neural network that computes $\alpha_j(t)$, i.e., the forward probability of an HMM producing the partial sequence y_1^t and ending up in state j, so that isolated words can be recognized by comparing their final scores $\alpha_F(T)$. Figure 4.5 motivates the construction of an AlphaNet. The first panel illustrates the basic recurrence, $\alpha_j(t) = \sum_i \alpha_i(t-1)\, a_{ij}$. The second panel shows how this recurrence may be implemented using a recurrent network. The third panel shows how the additional term $b_j(y_t)$ can be factored into the equation, using sigma-pi units, so that the AlphaNet properly computes $\alpha_j(t) = \sum_i \alpha_i(t-1)\, a_{ij} b_j(y_t)$.

4.3.2. Frame Level Training

Rather than simply reimplementing an HMM using neural networks, most researchers have been exploring ways to enhance HMMs by designing hybrid systems that capitalize on the respective strengths of each technology: temporal modeling in the HMM and acous-

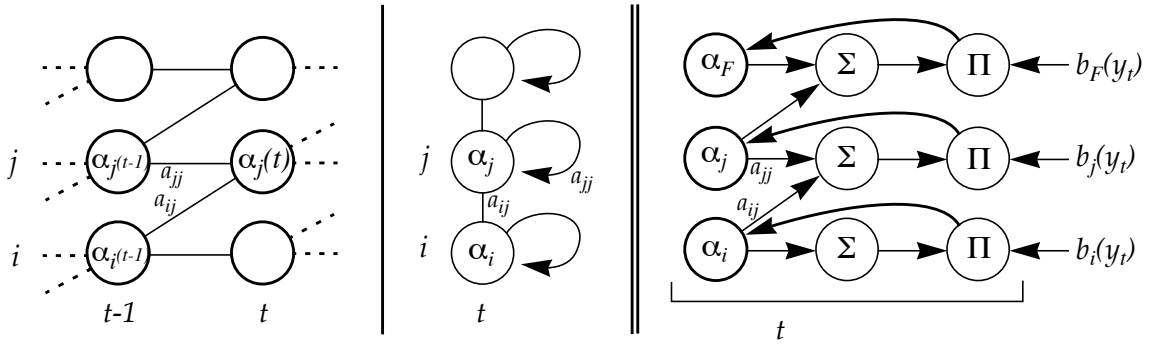


Figure 4.5: Construction of an AlphaNet (final panel).

tic modeling in neural networks. In particular, neural networks are often trained to compute emission probabilities for HMMs. Neural networks are well suited to this mapping task, and they also have a theoretical advantage over HMMs, because unlike discrete density HMMs, they can accept continuous-valued inputs and hence don't suffer from quantization errors; and unlike continuous density HMMs, they don't make any dubious assumptions about the parametric shape of the density function. There are many ways to design and train a neural network for this purpose. The simplest is to map frame inputs directly to emission symbol outputs, and to train such a network on a frame-by-frame basis. This approach is called *Frame Level Training*.

Frame level training has been extensively studied by researchers at Philips, ICSI, and SRI. Initial work by Bourlard and Wellekens (1988=1990) focused on the theoretical links between Hidden Markov Models and neural networks, establishing that neural networks estimate posterior probabilities which should be divided by priors in order to yield likelihoods for use in an HMM. Subsequent work at ICSI and SRI (Morgan & Bourlard 1990, Renals et al 1992, Bourlard & Morgan 1994) confirmed this insight in a series of experiments leading to excellent results on the Resource Management database. The simple MLPs in these experiments typically used an input window of 9 speech frames, 69 phoneme output units, and hundreds or even thousands of hidden units (taking advantage of the fact that more hidden units always gave better results); a parallel computer was used to train millions of weights in a reasonable amount of time. Good results depended on careful use of the neural networks, with techniques that included online training, random sampling of the training data, cross-validation, step size adaptation, heuristic bias initialization, and division by priors during recognition. A baseline system achieved 12.8% word error on the RM database using speaker-independent phoneme models; this improved to 8.3% by adding multiple pronunciations and cross-word modeling, and further improved to 7.9% by interpolating the likelihoods obtained from the MLP with those from SRI's DECIPHER system (which obtained 14.0% by itself under similar conditions). Finally, it was demonstrated that when using the same number of parameters, an MLP can outperform an HMM (e.g., achieving 8.3% vs 11.0% word error with 150,000 parameters), because an MLP makes fewer questionable assumptions about the parameter space.

Franzini, Lee, & Waibel (1990) have also studied frame level training. They started with an HMM, whose emission probabilities were represented by a histogram over a VQ codebook, and replaced this mechanism by a neural network that served the same purpose; the targets for this network were continuous probabilities, rather than binary classes as used by Bourlard and his colleagues. The network's input was a window containing seven frames of speech (70 msec), and there was an output unit for each probability distribution to be modeled[1]. Their network also had two hidden layers, the first of which was recurrent, via a buffer of the past 10 copies of the hidden layer which was fed back into that same hidden layer, in a variation of the Elman Network architecture. (This buffer actually represented 500 msec of history, because the input window was advanced 5 frames, or 50 msec, at a time.) The system was evaluated on the TI/NBS Speaker-Independent Continuous Digits Database, and achieved 98.5% word recognition accuracy, close to the best known result of 99.5%.

4.3.3. Segment Level Training

An alternative to frame-level training is segment-level training, in which a neural network receives input from an entire segment of speech (e.g., the whole duration of a phoneme), rather than from a single frame or a fixed window of frames. This allows the network to take better advantage of the correlation that exists among all the frames of the segment, and also makes it easier to incorporate segmental information, such as duration. The drawback of this approach is that the speech must first be segmented before the neural network can evaluate the segments.

The TDNN (Waibel et al 1989) represented an early attempt at segment-level training, as its output units were designed to integrate partial evidence from the whole duration of a phoneme, so that the network was purportedly trained at the phoneme level rather than at the frame level. However, the TDNN's input window assumed a constant width of 15 frames for all phonemes, so it did not truly operate at the segment level; and this architecture was only applied to phoneme recognition, not word recognition.

Austin et al (1992) at BBN explored true segment-level training for large vocabulary continuous speech recognition. A *Segmental Neural Network* (SNN) was trained to classify phonemes from variable-duration segments of speech; the variable-duration segments were linearly downsampled to a uniform width of five frames for the SNN. All phonemic segmentations were provided by a state-of-the-art HMM system. During training, the SNN was taught to correctly classify each segment of each utterance. During testing, the SNN was given the segmentations of the N-best sentence hypotheses from the HMM; the SNN produced a composite score for each sentence (the product of the scores and the duration probabilities[2] of all segments), and these SNN scores and HMM scores were combined to identify the single best sentence. This system achieved 11.6% word error on the RM database. Later, performance improved to 9.0% error when the SNN was also trained negatively

1. In this HMM, output symbols were emitted during transitions rather than in states, so there was actually one output unit per transition rather than per state.
2. Duration probabilities were provided by a smoothed histogram over all durations obtained from the training data.

on incorrect segments from N-best sentence hypotheses, thus preparing the system for the kinds of confusions that it was likely to encounter in N-best lists during testing.

4.3.4. Word Level Training

A natural extension to segment-level training is word-level training, in which a neural network receives input from an entire word, and is directly trained to optimize word classification accuracy. Word level training is appealing because it brings the training criterion still closer to the ultimate testing criterion of sentence recognition accuracy. Unfortunately the extension is nontrivial, because in contrast to a simple phoneme, a word cannot be adequately modeled by a single state, but requires a sequence of states; and the activations of these states cannot be simply summed over time as in a TDNN, but must first be segmented by a dynamic time warping procedure (DTW), identifying which states apply to which frames. Thus, word-level training requires that DTW be embedded into a neural network.

This was first achieved by Sakoe et al (1989), in an architecture called the *Dynamic programming Neural Network* (DNN). The DNN is a network in which the hidden units represent states, and the output units represent words. For each word unit, an alignment path between its states and the inputs is established by DTW, and the output unit integrates the activations of the hidden units (states) along the alignment path. The network is trained to output 1 for the correct word unit, and 0 for all incorrect word units. The DTW alignment path may be static (established before training begins) or dynamic (reestablished during each iteration of training); static alignment is obviously more efficient, but dynamic alignment was shown to give better results. The DNN was applied to a Japanese database of isolated digits, and achieved 99.3% word accuracy, outperforming pure DTW (98.9%).

Haffner (1991) similarly incorporated DTW into the high-performance TDNN architecture, yielding the *Multi-State Time Delay Neural Network* (MS-TDNN), as illustrated in Figure 4.6. In contrast to Sakoe's system, the MS-TDNN has an extra hidden layer and a hierarchy of time delays, so that it may form more powerful feature detectors; and its DTW path accumulates one score per frame rather than one score per state, so it is more easily extended to continuous speech recognition (Ney 1984). The MS-TDNN was applied to a database of spoken letters, and achieved an average of 93.6% word accuracy, compared to 90.0% for Sphinx[1]. The MS-TDNN benefitted from some novel techniques, including "transition states" between adjacent phonemic states (e.g., `B-IY` between the `B` and `IY` states, set to a linear combination of the activations of `B` and `IY`), and specially trained "boundary detection units" (BDU), which allowed word transitions only when the BDU activation exceeded a threshold value.

Hild and Waibel (1993) improved on Haffner's MS-TDNN, achieving 94.8% word accuracy on the same database of spoken letters, or 92.0% on the Resource Management spell mode database. Their improvements included (a) free alignment across word boundaries, i.e., using DTW on a segment of speech wider than the word to identify the word's boundaries dynamically during training; (b) word duration modeling, i.e., penalizing words by add-

1. In this comparison, Sphinx also had the advantage of using context-dependent phoneme models, while the MS-TDNN used context-independent models.

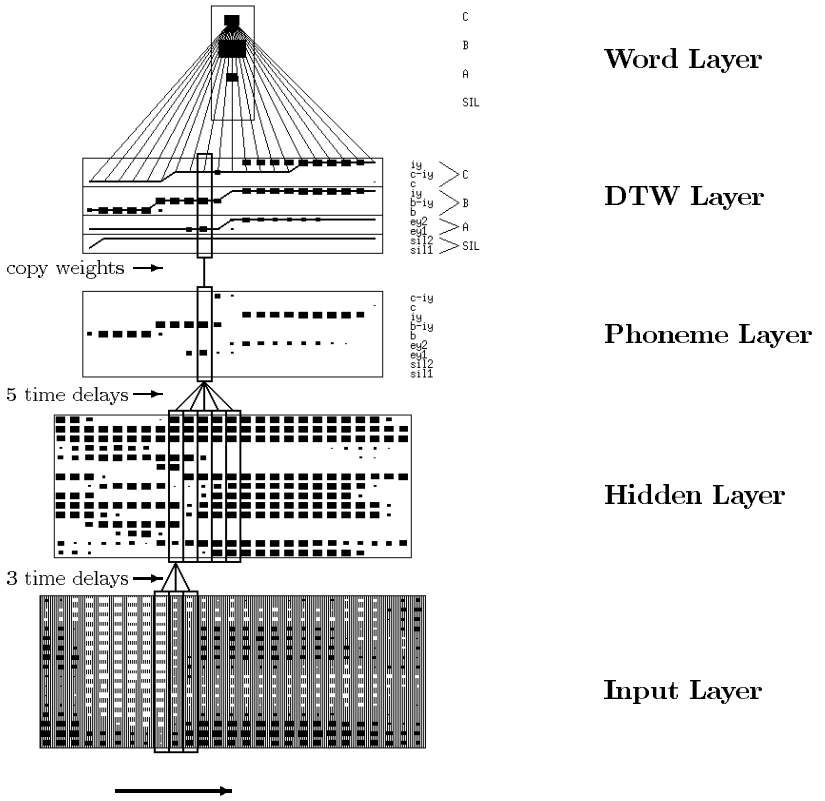


Figure 4.6: MS-TDNN recognizing the word "B". Only the activations for the words "SIL", "A", "B", and "C" are shown. (From Hild & Waibel, 1993).

ing the logarithm of their duration probabilities, derived from a histogram and scaled by a factor that balances insertions and deletions; and (c) sentence level training, i.e., training positively on the correct alignment path and training negatively on incorrect parts of an alignment path that is obtained by testing.

Tebelskis (1993) applied the MS-TDNN to large vocabulary continuous speech recognition. This work is detailed later in this thesis.

4.3.5. Global Optimization

The trend in NN-HMM hybrids has been towards global optimization of system parameters, i.e., relaxing the rigidities in a system so its performance is less handicapped by false assumptions. Segment-level training and word-level training are two important steps towards global optimization, as they bypass the rigid assumption that frame accuracy is correlated with word accuracy, making the training criterion more consistent with the testing criterion.

Another step towards global optimization, pursued by Bengio et al (1992), is the joint optimization of the input representation with the rest of the system. Bengio proposed a NN-HMM hybrid in which the speech frames are produced by a combination of signal analysis

and neural networks; the speech frames then serve as inputs for an ordinary HMM. The neural networks are trained to produce increasingly useful speech frames, by backpropagating an error gradient that derives from the HMM's own optimization criterion, so that the neural networks and the HMM are optimized simultaneously. This technique was evaluated on the task of speaker independent plosive recognition, i.e., distinguishing between the phonemes /b,d,g,p,t,k,dx,other/. When the HMM was trained separately from the neural networks, recognition accuracy was only 75%; but when it was trained with global optimization, recognition accuracy jumped to 86%.

4.3.6. Context Dependence

It is well known that the accuracy of an HMM improves with the context sensitivity of its acoustic models. In particular, context dependent models (such as triphones) perform better than context independent models (such as phonemes). This has led researchers to try to improve the accuracy of hybrid NN-HMM systems by likewise making them more context sensitive. Four ways to achieve this are illustrated in Figure 4.7.

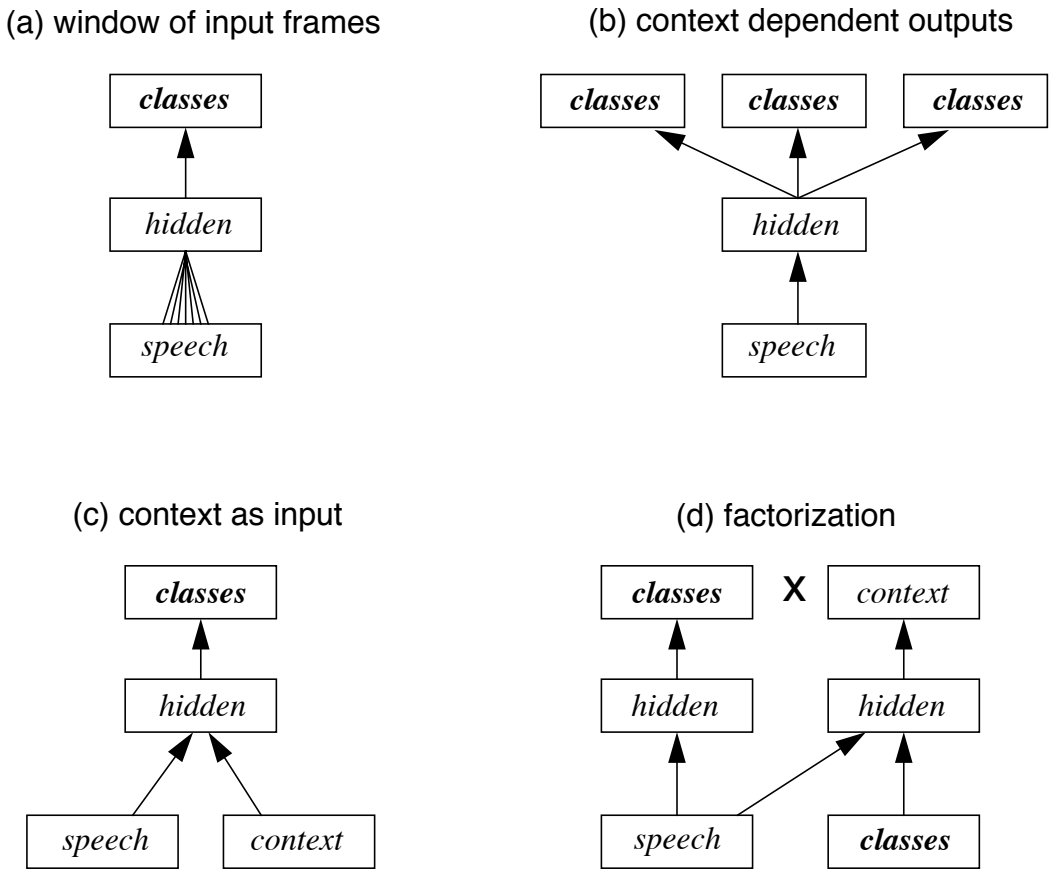


Figure 4.7: Four approaches to context dependent modeling.

The first technique is simply to provide a window of speech frames, rather than a single frame, as input to the network. The arbitrary width of the input window is constrained only by computational requirements and the diminishing relevance of distant frames. This technique is so trivial and useful for a neural network that it is used in virtually all NN-HMM hybrids; it can also be used in combination with the remaining techniques in this section. By contrast, in a standard HMM, the Independence Assumption prevents the system from taking advantage of neighboring frames directly. The only way an HMM can exploit the correlation between neighboring frames is by artificially absorbing them into the current frame (e.g., by defining multiple simultaneous streams of data to impart the frames and/or their deltas, or by using LDA to transform these streams into a single stream).

A window of input frames provides context sensitivity, but not context dependence. Context dependence implies that there is a separate model for each context, e.g., a model for /A/ when embedded in the context "kab", a separate model for /A/ when embedded in the context "tap", etc. The following techniques support true context dependent modeling in NN-HMM hybrids.

In technique (b), the most naive approach, there is a separate output unit for each context-dependent model. For example, if there are 50 phonemes, then it will require 50x50 = 2500 outputs in order to model diphones (phonemes in the context of their immediate neighbor), or 50x50x50 = 125000 outputs to model triphones (phonemes in the context of both their left and right neighbor). An obvious problem with this approach, shared by analogous HMMs, is that there is unlikely to be enough training data to adequately train all of the parameters of the system. Consequently, this approach has rarely been used in practice.

A more economical approach (c) is to use a single network that accepts a description of the context as part of its input, as suggested by Petek et al (1991). Left-phoneme context dependence, for example, could be implemented by a boolean localist representation of the left phoneme; or, more compactly, by a binary encoding of its linguistic features, or by its principal components discovered automatically by an encoder network. Note that in order to model triphones instead of diphones, we only need to double the number of context units, rather than using 50 times as many models. Training is efficient because the full context is available in the training sentences; however, testing may require many forward passes with different contextual inputs, because the context is not always known. Petek showed that

Figure 4.8: Contextual inputs. Left: standard implementation. Right: efficient implementation.

these forward passes can be made more efficient by heuristically splitting the hidden layer as shown in Figure 4.8, such that the speech and the context feed into independent parts, and each context effectively contributes a different bias to the output units; after training is complete, these contextual output biases can be precomputed, reducing the family of forward passes to a family of output sigmoid computations. Contextual inputs helped to increase the absolute word accuracy of Petek's system from 60% to 72%.

Bourlard et al (1992) proposed a fourth approach to context dependence, based on factorization (d). When a neural network is trained as a phoneme classifier, it estimates $P(q|x)$, where q is the phoneme class and x is the speech input. To introduce context dependence, we would like to estimate $P(q,c|x)$, where c is the phonetic context. This can be decomposed as follows:

$$P(q, c|x) = P(q|x) \cdot P(c|q,x) \tag{56}$$

This says that the context dependent probability is equal to the product of two terms: $P(q|x)$ which is the output activation of a standard network, and $P(c|q,x)$ which is the output activation of an auxiliary network whose inputs are speech as well as the current phoneme class, and whose outputs range over the contextual phoneme classes, as illustrated in Figure 4.7(d). The resulting context dependent posterior can then be converted to a likelihood by Bayes Rule:

$$P(x|q,c) = \frac{P(q, c|x) \cdot P(x)}{P(q, c)} \tag{57}$$

where $P(x)$ can be ignored during recognition because it's a constant in each frame, and the prior $P(q,c)$ can be evaluated directly from the training set.

This factorization approach can easily be extended to triphone modeling. For triphones, we want to estimate $P(q,c_l,c_r|x)$, where c_l is the left phonetic context and c_r is right phonetic context. This can be decomposed as follows:

$$P(q, c_l, c_r|x) = P(q|x) \cdot P(c_l|q,x) \cdot P(c_r|c_l,q,x) \tag{58}$$

Similarly,

$$P(q, c_l,c_r) = P(q) \cdot P(c_l|q) \cdot P(c_r|c_l,q) \tag{59}$$

These six terms can be estimated by neural networks whose inputs and outputs correspond to each i and o in $P(o|i)$; in fact some of the terms in Equation (59) are so simple that they can be evaluated directly from the training data. The posterior in Equation (58) can be converted to a likelihood by Bayes Rule:

$$P(x|q,c_l,c_r) = \frac{P(q, c_l,c_r|x) \cdot P(x)}{P(q, c_l,c_r)} \tag{60}$$

$$= \frac{P(q|x) \cdot P(c_l|q,x) \cdot P(c_r|c_l,q,x)}{P(q) \cdot P(c_l|q) \cdot P(c_r|c_l,q)} \cdot P(x) \quad (61)$$

where $P(x)$ can again be ignored during recognition, and the other six terms can be taken from the outputs of the six neural networks. This likelihood can be used for Viterbi alignment.

As in approach (c), a family of forward passes during recognition can be reduced to a family of output sigmoid computations, by splitting the hidden layer and caching the effective output biases from the contextual inputs. Preliminary experiments showed that splitting the hidden layer in this way did not degrade the accuracy of a network, and triphone models were rendered only 2-3 times slower than monophone models.

4.3.7. Speaker Independence

Experience with HMMs has shown that speaker independent systems typically make 2-3 times as many errors as speaker dependent systems (Lee 1988), simply because there is greater variability between speakers than within a single speaker. HMMs typically deal with this problem by merely increasing the number of context-dependent models, in the hope of better covering the variabilities between speakers.

NN-HMM hybrids suffer from a similar gap in performance between speaker dependence and speaker independence. For example, Schmidbauer and Tebelskis (1992), using an LVQ-based hybrid, obtained an average of 14% error on speaker-dependent data, versus 32% error when the same network was applied to speaker-independent data. Several techniques aimed at closing this gap have been developed for NN-HMM hybrids. Figure 4.9 illustrates the baseline approach of training a standard network on the data from all speakers (panel a), followed by three improvements upon this (b,c,d).

The first improvement, shown as technique (b), is a **mixture of speaker-dependent models**, resembling the Mixture of Experts paradigm promoted by Jacobs et al (1991). In this approach, several networks are trained independently on data from different speakers, while a "speaker ID" network is trained to identify the corresponding speaker; during recognition, speech is presented to all networks in parallel, and the outputs of the speaker ID network specify a linear combination of the speaker-dependent networks, to yield an overall result. This approach makes it easier to classify phones correctly, because it separates and hence reduces the overlap of distributions that come from different speakers. It also yields multi-speaker accuracy[1] close to speaker-dependent accuracy, the only source of degradation being imperfect speaker identification. Among the researchers who have studied this approach:

- Hampshire and Waibel (1990) first used this approach in their *Meta-Pi network*, which consisted of six speaker-dependent TDNNs plus a speaker ID network containing one unit per TDNN, all trained by backpropagation. This network obtained 98.4% phoneme accuracy in multi-speaker mode, significantly outperforming a

1. "Multi-speaker" evaluation means testing on speakers who were in the training set.

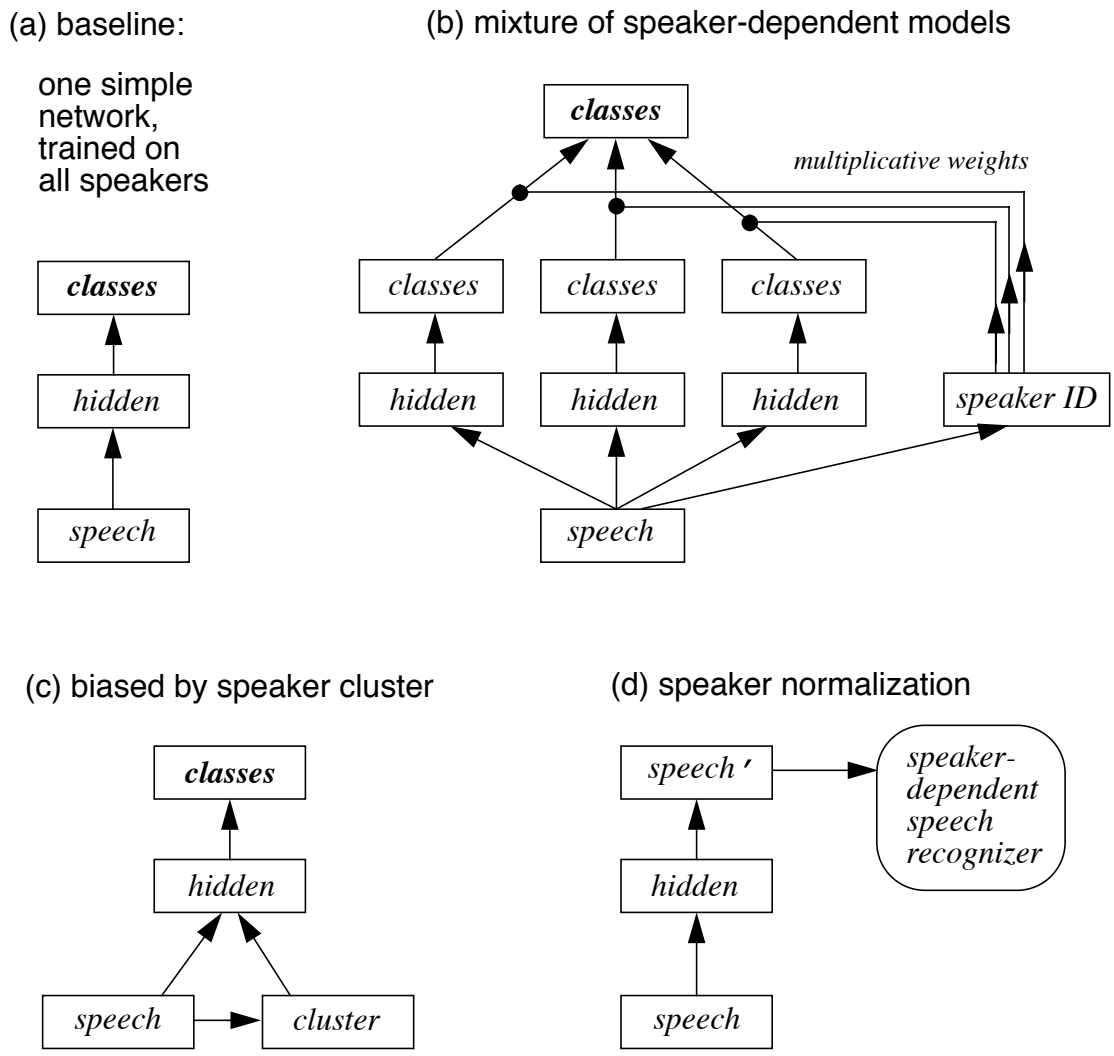


Figure 4.9: Four approaches to speaker independent modeling.

baseline TDNN which obtained only 95.9% accuracy. Remarkably, one of the speakers (MHT) obtained 99.8% phoneme accuracy, even though the speaker ID network failed to recognize him and thus ignored the outputs of MHT's own TDNN network, because the system had formed a robust linear combination of other speakers whom he resembled.

- Kubala and Schwartz (1991) adapted this approach to a standard HMM system, mixing their speaker-dependent HMMs with fixed weights instead of a speaker ID network. They found that only 12 speaker-dependent HMMs were needed in order to attain the same word recognition accuracy as a baseline system trained on 109 speakers (using a comparable amount of total data in each case). Because of this, and because it's cheaper to collect a large amount of data from a few speakers than

to collect a small amount of data from many speakers, Kubala and Schwartz concluded that this technique is also valuable for reducing the cost of data collection.

- Schmidbauer and Tebelskis (1992) incorporated this approach into an LVQ-HMM hybrid for continuous speech recognition. Four speaker-*biased* phoneme models (for pooled males, pooled females, and two individuals) were mixed using a correspondingly generalized speaker ID network, whose activations for the 40 separate phonemes were established using five "rapid adaptation" sentences. The rapid adaptation bought only a small improvement over speaker-independent results (59% vs. 55% word accuracy), perhaps because there were so few speaker-biased models in the system. Long-term adaptation, in which all system parameters received additional training on correctly recognized test sentences, resulted in a greater improvement (to 73%), although still falling short of speaker-dependent accuracy (82%).
- Hild and Waibel (1993) performed a battery of experiments with MS-TDNNs on spelled letter recognition, to determine the best level of speaker and parameter specificity for their networks, as well as the best way to mix the networks together. They found that segregating the speakers is always better than pooling everyone together, although some degree of parameter sharing between the segregated networks is often helpful (given limited training data). In particular, it was often best to mix only their lower layers, and to use shared structure at higher layers. They also found that mixing the networks according to the results of a brief adaptation phase (as in Schmidbauer and Tebelskis) is generally more effective than using an instantaneous speaker ID network, although the latter technique gives comparable results in multi-speaker testing. Applying their best techniques to the speaker-independent Resource Management spell mode database, they obtained 92.0% word accuracy, outperforming Sphinx (90.4%).

Another way to improve speaker-independent accuracy is to **bias the network** using extra inputs that characterize the speaker, as shown in Figure 4.9(c). The extra inputs are determined automatically from the input speech, hence they represent some sort of cluster to which the speaker belongs. Like the Mixture of Experts approach, this technique improves phoneme classification accuracy by separating the distributions of different speakers, reducing their overlap and hence their confusability. It has the additional advantage of adapting very quickly to a new speaker's voice, typically requiring only a few words rather than several whole sentences. Among the researchers who have studied this approach:

- Witbrock and Haffner (1992) developed the *Speaker Voice Code network* (SVC-net), a system that learns to quickly identify where a speaker's voice lies in a space of possible voices. An SVC is a 2 unit code, derived as the bottleneck of an encoder network that is trained to reproduce a speaker's complete set of phoneme pronunciation codes (PPCs), each of which is a 3-unit code that was likewise derived as the bottleneck of an encoder network that was trained to reproduce the acoustic patterns associated with that particular phoneme. The SVC code varied considerably between speakers, yet proved remarkably stable for any given speaker, regardless of the phonemes that were available for its estimation in only a few words of speech. When the SVC code was provided as an extra input to an

MS-TDNN, the word accuracy on a digit recognition task improved from 1.10% error to 0.99% error.

- Konig and Morgan (1993) experimented with the *Speaker Cluster Neural Network* (SCNN), a continuous speech recognizer in which an MLP's inputs were supplemented by a small number of binary units describing the speaker cluster. When two such inputs were used, representing the speaker's gender (as determined with 98.3% accuracy by a neural network that had received supervised training), performance on the Resource Management database improved from 10.6% error to 10.2% error. Alternatively, when speakers were clustered in an unsupervised fashion, by applying k-means clustering to the acoustic centroids of each speaker (for k = 2 through 5 clusters), performance improved to an intermediate level of 10.4% error.

A final way to improve speaker-independent accuracy is through **speaker normalization**, as shown in Figure 4.9(d). In this approach, one speaker is designated as the reference speaker, and a speaker-dependent system is trained to high accuracy on his voice; then, in order to recognize speech from a new speaker (say, a female), her acoustic frames are mapped by a neural network into corresponding frames in the reference speaker's voice, which can then be fed into the speaker-dependent system.

- Huang (1992a) explored speaker normalization, using a conventional HMM for speaker-dependent recognition (achieving 1.4% word error on the reference speaker), and a simple MLP for nonlinear frame normalization. This normalization network was trained on 40 adaptation sentences for each new speaker, using DTW to establish the correspondence between input frames (from the new speaker) and output frames (for the reference speaker). The system was evaluated on the speaker-dependent portion of the Resource Management database; impressively, speaker normalization reduced the cross-speaker error rate from 41.9% error to 6.8% error. The error rate was further reduced to 5.0% by using eight codeword-dependent neural networks instead of a single monolithic network, as the task of each network was considerably simplified. This final error rate is comparable to the error rate of speaker-independent systems on this database; hence Huang concluded that speaker normalization can be useful in situations where large amounts of training data are available only for one speaker and you want to recognize other people's speech.

4.3.8. Word Spotting

Continuous speech recognition normally assumes that every spoken word should be correctly recognized. However, there are some applications where in fact only very few vocabulary words (called *keywords*) carry any significance, and the rest of an utterance can be ignored. For example, a system might prompt the user with a question, and then only listen for the words "yes" or "no", which may be embedded within a long response. For such applications, a *word spotter*, which listens for and flags only these keywords, may be more useful than a full-blown continuous speech recognition system. Several researchers have

recently designed word spotting systems that incorporate both neural networks and HMMs. Among these systems, there have been two basic strategies for deploying a neural network:

1. A neural network may serve as a secondary system that reevaluates the putative hits identified by a primary HMM system. In this case, the network's architecture can be rather simple, because an already-detected keyword candidate can easily be normalized to a fixed duration for the network's input.

2. A neural network may serve as the primary word spotter. In this case, the network's architecture must be more complex, because it must automatically warp the utterance while it scans for keywords.

David Morgan et al (1991) explored the first strategy, using a primary word spotter that was based on DTW rather than HMMs. When this system detected a keyword candidate, its speech frames were converted to a fixed-length representation (using either a Fourier transform, a linear compression of the speech frames, a network-generated compression, or a combination of these); and then this fixed-length representation was reevaluated by an appropriately trained neural network (either an RCE network[1], a probabilistic RCE network, or a modularized hierarchy of these), so that the network could decide whether to reject the candidate as a "false alarm". This system was evaluated on the "Stonehenge X" database. One rather arcane combination of the above techniques eliminated 72% of the false alarms generated by the primary system, while only rejecting 2% of the true keywords (i.e., word spotting accuracy declined from 80% to 78%).

Zeppenfeld and Waibel (1992,1993) explored the second strategy, using an MS-TDNN as a primary word spotter. This system represented keywords with unlabeled state models rather than shared phoneme models, due to the coarseness of the database. The MS-TDNN produced a score for each keyword in every frame, derived from the keyword's best DTW score in a range of frames beginning in the current frame. The system was first bootstrapped with state-level training on a forced linear alignment within each keyword, and then trained with backpropagation from the word level; positive and negative training were carefully balanced in both phases. It achieved a Figure of Merit[2] of 82.5% on the Road Rally database. Subsequent improvements — which included adding noise to improve generalization, subtracting spectral averages to normalize different databases, using duration constraints, grouping and balancing the keywords by their frequency of occurrence, extending short keywords into their nearby context, and modeling variant suffixes — contributed to a Figure of Merit of 72.2% on the official Stonehenge database, or 50.9% on the official Switchboard database.

Lippmann and Singer (1993) explored both of the above strategies. First, they used a high-performance tied-mixture HMM as a primary word spotter, and a simple MLP as a secondary tester. Candidate keywords from the primary system were linearly normalized to a fixed width for the neural network. The network reduced the false alarm rate by 16.4% on the Stonehenge database. This network apparently suffered from a poverty of training data;

1. Restricted Coloumb Energy network. RCE is a trademark of Nestor, Inc.
2. Figure of Merit summarizes a tradeoff between detection rate and false alarm rate. It is computed as the average detection rate for system configurations that achieve between 0 and 10 false alarms per keyword per hour.

attempts were made to augment the training set with false alarms obtained from an independent database, but this failed to improve the system's performance because the databases were too different, and hence too easily discriminable. The second strategy was then explored, using a primary network closely resembling Zeppenfeld's MS-TDNN, except that the hidden layer used radial basis functions instead of sigmoidal units. This enabled new RBF units to be added dynamically, as their Gaussians could be automatically centered on false alarms that arose in training, to simplify the goal of avoiding such mistakes in the future.

4.4. Summary

The field of speech recognition has seen tremendous activity in recent years. Hidden Markov Models still dominate the field, but many researchers have begun to explore ways in which neural networks can enhance the accuracy of HMM-based systems. Researchers into NN-HMM hybrids have explored many techniques (e.g., frame level training, segment level training, word level training, global optimization), many issues (e.g., temporal modeling, parameter sharing, context dependence, speaker independence), and many tasks (e.g., isolated word recognition, continuous speech recognition, word spotting). These explorations have especially proliferated since 1990, when this thesis was proposed, hence it is not surprising that there is a great deal of overlap between this thesis and concurrent developments in the field. The remainder of this thesis will present the results of my own research in the area of NN-HMM hybrids.

5. Databases

We performed our experiments on NN-HMM hybrids using three different databases: ATR's database of isolated Japanese words, the CMU Conference Registration database, and the DARPA Resource Management database. In this chapter we will briefly describe each of these databases.

5.1. Japanese Isolated Words

Our very first experiments were performed using a database of 5240 isolated Japanese words (Sagisaka et al 1987), provided by ATR Interpreting Telephony Research Laboratory in Japan, with whom we were collaborating. This database includes recordings of all 5240 words by several different native Japanese speakers, all of whom are professional announcers; but our experiments used the data from only one male speaker (MAU). Each isolated word was recorded in a soundproof booth, and digitized at a 12 kHz sampling rate. A Hamming window and an FFT were applied to the input data to produce 16 melscale spectral coefficients every 10 msec.

Because our computational resources were limited at the time, we chose not to use all 5240 words in this database; instead, we extracted two subsets based on a limited number of phonemes:

- **Subset 1** = 299 words (representing 234 unique words, due to the presence of homophones), comprised of only the 7 phonemes a,i,u,o,k,s,sh (plus an eighth phoneme for silence). From these 299 words, we trained on 229 words, and tested on the remaining 70 words (of which 50 were homophones of training samples, and 20 were novel words). Table 5.1 shows this vocabulary.
- **Subset 2** =1078 words (representing 924 unique words), comprised of only the 13 phonemes a,i,u,e,o,k,r,s,t,kk,sh,ts,tt (plus a 14th phoneme for silence). From these 1078 words, we trained on 900 words, and tested on 178 words (of which 118 were homophones of training samples, and 60 were novel words).

Using homophones in the testing set allowed us to test generalization to new samples of known words, while the unique words allowed us to test generalization to novel words (i.e., vocabulary independence).

aa	ikou	ooku	kakoi	ku *	koushou	**sasai**	**shisso**	shousoku
ai	ishi **	oka	kakou *	kui	kousou	sasu **	shakai	shoku
aiso	**ishiki**	**okashii**	kasa	kuiki	kousoku	sasoi	shaku	shokki
au *	isha	okasu	kasai	kuu	kokuso	sasou	shako	su *
ao	ishou	oki	kashi	kuuki	koshi	sakka	shashou *	suisoku
aoi	isu	oku **	kashikoi	kuukou	koshou	sakkaku	shuu	suu *
aka *	ikka	okosu	kashu	kuusou	**koosu**	sakki	**shuui**	sukasu
akai	ikkou	oshii	kasu *	kuki	kosu *	sakku	shuukai	suki *
aki *	issai	oshoku	kasuka	**kusa**	kokka	sassou	shuukaku	suku
aku *	isshu	osu *	**kakki**	kusai	kokkai	sassoku	shuuki	sukuu *
akushu	issho	osoi	kakko	kushi *	kokkaku	shi **	shuusai	sukoshi
asa	isshou	osou	kakkou	ko	kokki	shiai	shuushuu	**sushi**
asai	isso	ka *	ki *	koi **	kokkou	shio *	shuushoku	susu
ashi	issou	kai **	kioku	koishii	sa	shikai *	shukusha	suso
asu	ukai	kaikaku	kikai **	kou *	saiku	shikaku *	shukushou	sou **
akka	uku	kaisai	kikaku *	koui *	saikou	shikashi	shusai	soui
asshuku	ushi	kaishi	kiki	kouka	kaishuu	shiki	shushi	souko
i	**usui**	**kaisha**	kiku **	koukai **	**saisho**	**shikisai**	shushoku	sousa *
ii	uso	kaishaku	kikou	koukou *	saisoku	shiku	shou *	sousaku *
iu	o	kaishou	kisaku	koukoku	sao	shikou *	shouka *	soushiki
ika	oi	kau *	kishi	kousa	saka	shisaku	shoukai	soushoku
iasu	oishii	kao	kisha	**kousai**	sakai	shishuu	shouki	soko
iki **	ou *	kaoku	kishou *	kousaku *	sakasa	shishou	**shouko**	soshi
ikiiki	ooi *	kaku ***	kisuu	koushi *	saki	shisou	shousai	soshiki
ikioi	oou	kakusu	kiso *	koushiki	saku ***	shikkaku	shoushou	**soshou**
iku	ookii	**kako**	**kisoku**	koushuu *	sakusha	shikki	shousuu	sosokkashii

Table 5.1: Japanese isolated word vocabulary (Subset 1 = 299 samples including homophones; 234 unique words). The testing set (70 words) consisted of 50 homophones (starred words) and 20 novel words (in bold).

5.2. Conference Registration

Our first experiments with continuous speech recognition were performed using an early version of the CMU Conference Registration database (Wood 1992). The database consists of 204 English sentences using a vocabulary of 402 words, comprising 12 hypothetical dialogs in the domain of conference registration. A typical dialog is shown in Table 5.2; both sides of the conversation are read by the same speaker. Training and testing versions of this database were recorded with a close-speaking microphone in a quiet office by multiple speakers for speaker-dependent experiments. Recordings were digitized at a sampling rate of 16 kHz; a Hamming window and an FFT were computed, to produce 16 melscale spectral coefficients every 10 msec.

Since there are 402 words in the vocabulary, this database has a perplexity[1] of 402 when testing without a grammar. Since recognition is very difficult under such conditions, we created a word pair grammar (indicating which words can follow which other words) from the textual corpus. Unfortunately, with a perplexity of only 7, this word pair grammar soon proved too easy — it's hard to identify significant improvements above 97% word accuracy.

1. Perplexity is a measure of the branching factor in the grammar, i.e., the number of words that can follow any given word.

A: Hello, is this the office for the conference?
B: Yes, that's right.
A: I would like to register for the conference.
B: Do you already have a registration form?
A: No, not yet.
B: I see. Then I'll send you a registration form.
B: Could you give me your name and address?
A: The address is five thousand Forbes Avenue, Pittsburgh, Pennsylvania, one five two three six.
A: The name is David Johnson.
B: I see. I'll send you a registration form immediately.
B: If there are any questions, please ask me at any time.
A: Thank you. Goodbye.
B: Goodbye.

Table 5.2: A typical dialog in the Conference Registration database.

Therefore, we usually evaluated recognition accuracy at a perplexity of 111, by testing only the first three dialogs (41 sentences) using a reduced vocabulary without a grammar.

The Conference Registration database was developed in conjunction with the Janus Speech-to-Speech Translation system at CMU (Waibel et al 1991, Osterholtz et al 1992, Woszczyna et al 1994). While a full discussion of Janus is beyond the scope of this thesis, it is worth mentioning here that Janus is designed to automatically translate between two spoken languages (e.g., English and Japanese), so that the above dialog could be carried out between an American who wants to register for a conference in Tokyo but who speaks no Japanese, and a Japanese receptionist who speaks no English. Janus performs speech translation by integrating three modules — speech recognition, text translation, and speech generation — into a single end-to-end system. Each of these modules can use any available technology, and in fact various combinations of connectionist, stochastic, and/or symbolic approaches have been compared over the years. The speech recognition module, for example, was originally implemented by our LPNN, described in Chapter 6 (Waibel et al 1991, Osterholtz et al 1992); but it was later replaced by an LVQ-based speech recognizer with higher accuracy. Most recently, Janus has been expanded to a wide range of source and destination languages (English, Japanese, German, Spanish, Korean, etc.); its task has broadened from simple read speech to arbitrary spontaneous speech; and its domain has changed from conference registration to appointment scheduling (Woszczyna et al 1994).

5.3. Resource Management

In order to fairly compare our results against those of researchers outside of CMU, we also ran experiments on the DARPA speaker-independent Resource Management database (Price et al 1988). This is a standard database consisting of 3990 training sentences in the domain of naval resource management, recorded by 109 speakers contributing roughly 36 sentences each; this training set has been supplemented by periodic releases of speaker-independent testing data over the years, for comparative evaluations. Some typical sentences are listed

in Table 5.3. The vocabulary consists of 997 words, many of which are easily confusable, such as what/what's/was, four/fourth, any/many, etc., as well as the singular, plural, and possessive forms of many nouns, and an abundance of function words (a, the, of, on, etc.) which are unstressed and poorly articulated. During testing, we normally used a word pair grammar[1], with a perplexity of 60.

ARE THERE TWO CARRIERS IN YELLOW SEA WITH TRAINING RATING MORE THAN C1
HOW MANY NUCLEAR SURFACE SHIPS ARE WITHIN FIFTY NINE MILES OF CONIFER
SET UNIT OF MEASURE TO METRIC
DRAW THE TRACK OF MISHAWAKA
WHAT IS COPELAND'S FUEL LEVEL AND FUEL CAPACITY
WHAT WAS ARKANSAS'S READINESS THE TWENTY NINTH OF JUNE
ADD AN AREA
DOES SASSAFRAS HAVE THE LARGEST FUEL CAPACITY OF ALL SIBERIAN SEA SUBMARINES
WAS MONDAY'S LAST HFDF SENSOR LOCATION FOR THE HAWKBILL IN MOZAMBIQUE CHANNEL
DO ANY SHIPS THAT ARE IN BASS STRAIT HAVE MORE FUEL THAN HER
EDIT THE ALERT INVOLVING AJAX
WHAT SHIPS WENT TO C2 ON EQUIPMENT AFTER TWELVE JULY
WILL THE EISENHOWER'S EQUIPMENT PROBLEM BE FIXED BY TWENTY THREE JANUARY
WHEN DID SHERMAN LAST DOWNGRADE FOR ASUW MISSION AREA
REDRAW FIJI IN LOW RESOLUTION
CLEAR ALL DATA SCREENS
HOW MANY LAMPS CRUISERS ARE IN MOZAMBIQUE CHANNEL
CLEAR THE DISPLAY
WHAT WAS PIGEON'S LOCATION AND ASUW AREA MISSION CODE TWENTY FIVE DECEMBER
DIDN'T ENGLAND ARRIVE AT MANCHESTER YESTERDAY

Table 5.3: Typical sentences from the Resource Management database.

From the training set of 3990 sentences, we normally used 3600 for actual training, and 390 (from other speakers) for cross validation. However, when we performed gender-dependent training, we further subdivided the database into males, with 2590 training and 240 cross validation sentences, and females, with 1060 training and 100 cross validation sentences. The cross validation sentences were used during development, in parallel with the training sentences. Official evaluations were performed using a reserved set of 600 test sentences (390 male and 210 female), representing the union of the Feb89 and Oct89 releases of testing data, contributed by 30 independent speakers.

1. Actually a word-class pair grammar, as all sentences in this database were generated by expanding templates based on word classes.

6. Predictive Networks

Neural networks can be trained to compute smooth, nonlinear, nonparametric functions from any input space to any output space. Two very general types of functions are *prediction* and *classification*, as shown in Figure 6.1. In a predictive network, the inputs are several frames of speech, and the outputs are a prediction of the next frame of speech; by using multiple predictive networks, one for each phone, their prediction errors can be compared, and the one with the least prediction error is considered the best match for that segment of speech. By contrast, in a classification network, the inputs are again several frames of speech, but the outputs directly classify the speech segment into one of the given classes.

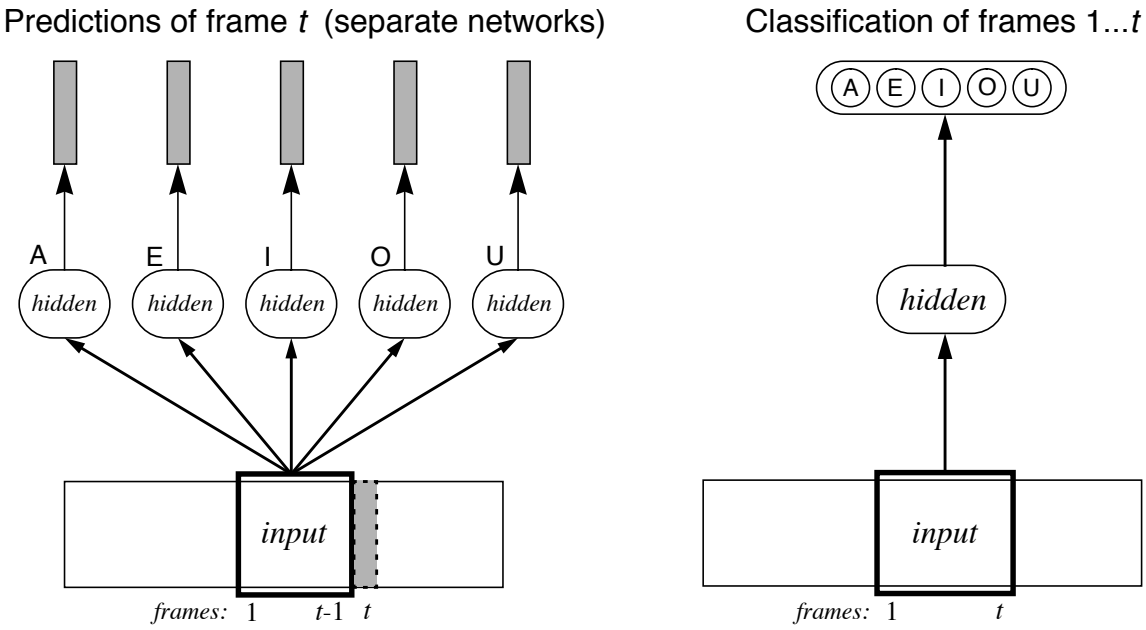


Figure 6.1: Prediction versus Classification.

In the course of our research, we have investigated both of these approaches. Predictive networks will be treated in this chapter, and classification networks will be treated in the next chapter.

6.1. Motivation... and Hindsight

We initially chose to explore predictive networks for a number of reasons. The principal reason was scientific curiosity — all of our colleagues in 1989 were studying classification networks, and we hoped that our novel approach might yield new insights and improved results. On a technical level, we argued that:

1. Classification networks are trained on binary output targets, and therefore they produce quasi-binary outputs, which are nontrivial to integrate into a speech recognition system because binary phoneme-level errors tend to confound word-level hypotheses. By contrast, predictive networks provide a simple way to get non-binary acoustic scores (prediction errors), with straightforward integration into a speech recognition system.
2. The temporal correlation between adjacent frames of speech is explicitly modeled by the predictive approach, but not by the classification approach. Thus, predictive networks offer a dynamical systems approach to speech recognition (Tishby 1990).
3. Predictive networks are *nonlinear* models, which can presumably model the dynamic properties of speech (e.g., curvature) better than linear predictive models.
4. Classification networks yield only one output per class, while predictive networks yield a whole frame of coefficients per class, representing a more detailed acoustic model.
5. The predictive approach uses a separate, independent network for each phoneme class, while the classification approach uses one integrated network. Therefore:
    - With the predictive approach, new phoneme classes can be introduced and trained at any time without impacting the rest of the system. By contrast, if new classes are added to a classification network, the entire system must be retrained.
    - The predictive approach offers more potential for parallelism.

As we gained more experience with predictive networks, however, we gradually realized that each of the above arguments was flawed in some way:

1. The fact that classification networks are trained on binary targets does not imply that such networks yield binary outputs. In fact, in recent years it has become clear that classification networks yield estimates of the posterior probabilities $P(class|input)$, which can be integrated into an HMM more effectively than prediction distortion measures.
2. The temporal correlation between N adjacent frames of speech and the $N+1$st predicted frame is modeled just as well by a classification network that takes $N+1$ adjacent frames of speech as input. It does not matter whether temporal dynamics are modeled explicitly, as in a predictive network, or implicitly, as in a classifica-

tion network.

3. Nonlinearity is a feature of neural networks in general, hence this is not an advantage of predictive networks over classification networks.

4. Although predictive networks yield a whole frame of coefficients per class, these are quickly reduced to a single scalar value (the prediction error) — just as in a classification network. Furthermore, the modeling power of any network can be enhanced by simply adding more hidden units.

5. The fact that the predictive approach uses a separate, independent network for each phoneme class implies that there is no discrimination between classes, hence the predictive approach is inherently weaker than the classification approach. Moreover:

   - There is little practical value to being able to add new phoneme classes without retraining, because phoneme classes normally remain stable for years at a time, and when they are redesigned, the changes tend to be global in scope.
   - The fact that predictive networks have more potential for parallelism is irrelevant if they yield poor word recognition accuracy to begin with.

Unaware that our arguments for predictive networks were specious, we experimented with this approach for two years before concluding that predictive networks are a suboptimal approach to speech recognition. This chapter summarizes the work we performed.

6.2. Related Work

Predictive networks are closely related to a special class of HMMs known as an *autoregressive HMMs* (Rabiner 1989). In an autoregressive HMM, each state is associated not with an emission probability density function, but with an autoregressive function, which is assumed to predict the next frame as a function of some preceding frames, with some residual prediction error (or noise), i.e.:

$$x_t = F_k(X_{t-p}^{t-1}, \theta_k) + \varepsilon_{t,k} \tag{62}$$

where F_k is the autoregressive function for state k, X_{t-p}^{t-1} are the p frames before time t, θ_k are the trainable parameters of the function F_k, and $\varepsilon_{t,k}$ is the prediction error of state k at time t. It is further assumed that $\varepsilon_{t,k}$ is an independent and identically distributed (iid) random variable with probability density function $p_\varepsilon(\varepsilon|\lambda_k)$ with parameters λ_k and zero mean, typically represented by a gaussian distribution. It can be shown that

$$P(X_1^T, Q_1^T) \approx P(X_{p+1}^T, Q_{p+1}^T \,|\, X_1^p, Q_1^p)$$
$$= \prod_{t=p+1}^{T} p_\varepsilon(x_t - F_{k_t}(X_{t-p}^{t-1}, \theta_{k_t}) \,|\, \lambda_{k_t}) \cdot p(q_t | q_{t-1}) \tag{63}$$

This says that the likelihood of generating the utterance X_1^T along state path Q_1^T is approximated by the cumulative product of the prediction error probability (rather than the emission probability) and the transition probability, over all time frames. It can further be shown that during recognition, maximizing the joint likelihood $P(X_1^T, Q_1^T)$ is equivalent to minimizing the cumulative prediction error, which can be performed simply by applying standard DTW to the local prediction errors

$$\left\| x_t - F_k(X_{t-p}^{t-1}, \theta_k) \right\|^2 \tag{64}$$

Although autoregressive HMMs are theoretically attractive, they have never performed as well as standard HMMs (de La Noue et al 1989, Wellekens 1987), for reasons that remain unclear. Predictive networks might be expected to perform somewhat better than autoregressive HMMs, because they use nonlinear rather than linear prediction. Nevertheless, as will be shown, the performance of our predictive networks was likewise disappointing.

At the same time that we began our experiments, similar experiments were performed on a smaller scale by Iso & Watanabe (1990) and Levin (1990). Each of these researchers applied predictive networks to the simple task of digit recognition, with encouraging results. Iso & Watanabe used 10 word models composed of typically 11 states (i.e., predictors) per word; after training on five samples of each Japanese digit from 107 speakers, their system achieved 99.8% digit recognition accuracy (or 0.2% error) on testing data. They also confirmed that their nonlinear predictors outperformed linear predictors (0.9% error), as well as DTW with multiple templates (1.1% error).

Levin (1990) studied a variant of the predictive approach, called a *Hidden Control Neural Network*, in which all the states of a word were collapsed into a single predictor, modulated by an input signal that represented the state. Applying the HCNN to 8-state word models, she obtained 99.3% digit recognition accuracy on multi-speaker testing. Note that both Levin's experiments and Iso & Watanabe's experiments used non-shared models, as they focused on small vocabulary recognition. We also note that digit recognition is a particularly easy task.

In later work, Iso & Watanabe (1991) improved their system by the use of backward prediction, shared demisyllable models, and covariance matrices, with which they obtained 97.6% word accuracy on a speaker-dependent, isolated word, 5000 Japanese word recognition task. Mellouk and Gallinari (1993) addressed the discriminative problems of predictive networks; their work will be discussed later in this chapter.

6.3. Linked Predictive Neural Networks

We explored the use of predictive networks as acoustic models in an architecture that we called *Linked Predictive Neural Networks* (LPNN), which was designed for large vocabulary recognition of both isolated words and continuous speech. Since it was designed for large vocabulary recognition, it was based on shared phoneme models, i.e., phoneme models (represented by predictive neural networks) that were linked over different contexts — hence the name.

In this section we will describe the basic operation and training of the LPNN, followed by the experiments that we performed with isolated word recognition and continuous speech recognition.

6.3.1. Basic Operation

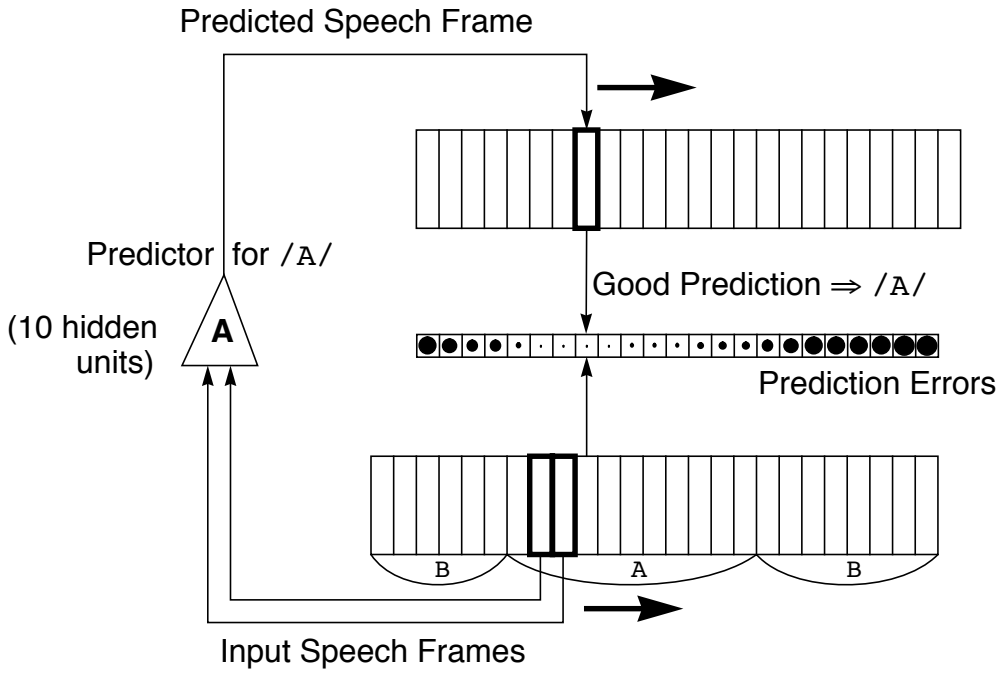


Figure 6.2: Basic operation of a predictive network.

An LPNN performs phoneme recognition via prediction, as shown in Figure 6.2(a). A network, shown as a triangle, takes K contiguous frames of speech (we normally used K=2), passes these through a hidden layer of units, and attempts to predict the next frame of speech. The predicted frame is then compared to the actual frame. If the error is small, the network is considered to be a good model for that segment of speech. If one could teach the network to make accurate predictions only during segments corresponding to the phoneme /A/ (for instance) and poor predictions elsewhere, then one would have an effective /A/ phoneme recognizer, by virtue of its contrast with other phoneme models. The LPNN satis-

fies this condition, by means of its training algorithm, so that we obtain a collection of phoneme recognizers, with one model per phoneme.

The LPNN is a NN-HMM hybrid, which means that acoustic modeling is performed by the predictive networks, while temporal modeling is performed by an HMM. This implies that the LPNN is a state-based system, such that each predictive network corresponds to a state in an (autoregressive) HMM. As in an HMM, phonemes can be modeled with finer granularity, using sub-phonetic state models. We normally used three states (predictive networks) per phoneme, as shown in subsequent diagrams. Also, as in an HMM, states (predictive networks) are sequenced hierarchically into words and sentences, following the constraints of a dictionary and a grammar.

6.3.2. Training the LPNN

Training the LPNN on an utterance proceeds in three steps: a forward pass, an alignment step, and a backward pass. The first two steps identify an optimal alignment between the acoustic models and the speech signal (if the utterance has been presegmented at the state level, then these two steps are unnecessary); this alignment is then used to force specialization in the acoustic models during the backward pass. We now describe the training algorithm in detail.

The first step is the forward pass, illustrated in Figure 6.3(a). For each frame of input speech at time t, we feed frame(t-1) and frame(t-2) in parallel into all the networks which are linked into this utterance, for example the networks a_1, a_2, a_3, b_1, b_2, and b_3 for the utterance "aba". Each network makes a prediction of frame(t), and its Euclidean distance from the actual frame(t) is computed. These scalar errors are broadcast and sequenced according to the known pronunciation of the utterance, and stored in column(t) of a prediction error matrix. This is repeated for each frame until the entire matrix has been computed.

The second step is the time alignment step, illustrated in Figure 6.3(b). The standard Dynamic Time Warping algorithm (DTW) is used to find an optimal alignment between the speech signal and the phoneme models, identified by a monotonically advancing diagonal path through the prediction error matrix, such that this path has the lowest possible cumulative error. The constraint of monotonicity ensures the proper sequencing of networks, corresponding to the progression of phonemes in the utterance.

The final step of training is the backward pass, illustrated in Figure 6.3(c). In this step, we backpropagate error at each point along the alignment path. In other words, for each frame we propagate error backwards into a single network, namely the one which best predicted that frame according to the alignment path; its backpropagated error is simply the difference between this network's prediction and the actual frame. A series of frames may backpropagate error into the same network, as shown. Error is accumulated in the networks until the last frame of the utterance, at which time all the weights are updated.

This completes the training for a single utterance. The same algorithm is repeated for all the utterances in the training set.

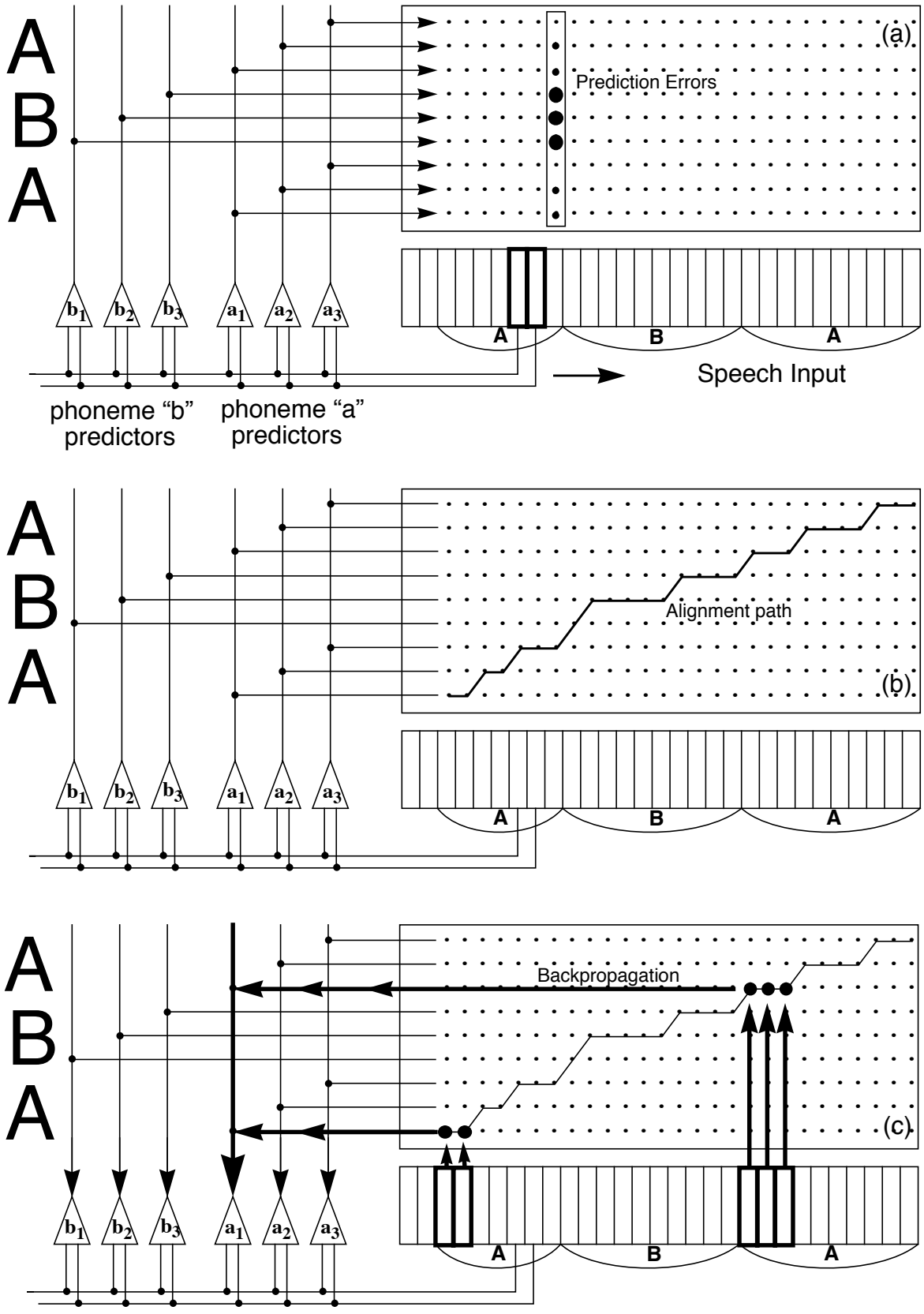


Figure 6.3: The LPNN training algorithm: (a) forward pass, (b) alignment, (c) backward pass.

It can be seen that by backpropagating error from different segments of speech into different networks, the networks learn to specialize on their associated segments of speech; consequently we obtain a full repertoire of individual phoneme models. This individuation in turn improves the accuracy of future alignments, in a self-correcting cycle. During the first iteration of training, when the weights have random values, it has proven useful to force an initial alignment based on average phoneme durations. During subsequent iterations, the LPNN itself segments the speech on the basis of the increasingly accurate alignments.

Testing is performed by applying standard DTW to the prediction errors for an unknown utterance. For isolated word recognition, this involves computing the DTW alignment path for all words in the vocabulary, and finding the word with the lowest score; if desired, next-best matches can be determined just by comparing scores. For continuous speech recognition, the One-Stage DTW algorithm (Ney 1984) is used to find the sequence of words with the lowest score; if desired, next-best sentences can be determined by using the N-best search algorithm (Schwartz and Chow 1990).

6.3.3. Isolated Word Recognition Experiments

We first evaluated the LPNN system on the task of isolated word recognition. While performing these experiments we explored a number of extensions to the basic LPNN system. Two simple extensions were quickly found to improve the system's performance, hence they were adopted as "standard" extensions, and used in all the experiments reported here.

The first standard extension was the use of duration constraints. We applied two types of duration constraints during recognition: 1) hard constraints, where any candidate word whose average duration differed by more than 20% from the given sample was rejected; and 2) soft constraints, where the optimal alignment score of a candidate word was penalized for discrepancies between the alignment-determined durations of its constituent phonemes and the known average duration of those same phonemes.

The second standard extension was a simple heuristic to sharpen word boundaries. For convenience, we include a "silence" phoneme in all our phoneme sets; this phoneme is linked in at the beginning and end of each isolated word, representing the background silence. Word boundaries were sharpened by artificially penalizing the prediction error for this "silence" phoneme whenever the signal exceeded the background noise level.

Our experiments were carried out on two different subsets of a Japanese database of isolated words, as described in Section 5.1. The first group contained almost 300 samples representing 234 unique words (limited to 8 particular phonemes), and the second contained 1078 samples representing 924 unique words (limited to 14 particular phonemes). Each of these groups was divided into training and testing sets; and the testing sets included both homophones of training samples (enabling us to test generalization to new samples of known words), and novel words (enabling us to test vocabulary independent generalization).

Our initial experiments on the 234 word vocabulary used a three-network model for each of the eight phonemes. After training for 200 iterations, recognition performance was perfect for the 20 novel words, and 45/50 (90%) correct for the homophones in the testing set. The fact that novel words were recognized better than new samples of familiar words is due

to the fact that most homophones are short confusable words (e.g., "kau" vs. "kao", or "kooshi" vs. "koshi"). By way of comparison, the recognition rate was 95% for the training set.

We then introduced further extensions to the system. The first of these was to allow a limited number of "alternate" models for each phoneme. Since phonemes have different characteristics in different contexts, the LPNN's phoneme modeling accuracy can be improved if independent networks are allocated for each type of context to be modeled. Alternates are thus analogous to context-dependent models. However, rather than assigning an explicit context for each alternate model, we let the system itself decide which alternate to use in a given context, by trying each alternate and linking in whichever one yields the lowest alignment score. When errors are backpropagated, the "winning" alternate is reinforced with backpropagated error in that context, while competing alternates remain unchanged.

We evaluated networks with as many as three alternate models per phoneme. As we expected, the alternates successfully distributed themselves over different contexts. For example, the three "k" alternates became specialized for the context of an initial "ki", other initial "k"s, and internal "k"s, respectively. We found that the addition of more alternates consistently improves performance on training data, as a result of crisper internal representations, but generalization to the test set eventually deteriorates as the amount of training data per alternate diminishes. The use of two alternates was generally found to be the best compromise between these competing factors.

Significant improvements were also obtained by expanding the set of phoneme models to explicitly represent consonants that in Japanese are only distinguishable by the duration of their stop closure (e.g., "k" versus "kk"). However, allocating new phoneme models to represent diphthongs (e.g., "au") did not improve results, presumably due to insufficient training data.

Table 6.1 shows the recognition performance of our two best LPNNs, for the 234 and 924 word vocabularies, respectively. Both of these LPNNs used all of the above optimizations. Their performance is shown for a range of ranks, where a rank of K means a word is considered correctly recognized if it appears among the best K candidates.

Vocab size	Rank	Testing set		Training set
		Homophones	Novel words	
234	1	47/50 (94%)	19/20 (95%)	228/229 (99%)
	2	49/50 (98%)	20/20 (100%)	229/229 (100%)
	3	50/50 (100%)	20/20 (100%)	229/229 (100%)
924	1	106/118 (90%)	55/60 (92%)	855/900 (95%)
	2	116/118 (98%)	58/60 (97%)	886/900 (98%)
	3	117/118 (99%)	60/60 (100%)	891/900 (99%)

Table 6.1: LPNN performance on isolated word recognition.

For the 234 word vocabulary, we achieved an overall recognition rate of 94% on test data using an exact match criterion, or 99% or 100% recognition within the top two or three candidates, respectively. For the 924 word vocabulary, our best results on the test data were 90% using an exact match criterion, or 97.7% or 99.4% recognition within the top two or three candidates, respectively. Among all the errors made for the 924 word vocabulary (on both training and testing sets), approximately 15% were due to duration problems, such as confusing "sei" and "seii"; another 12% were due to confusing "t" with "k", as in "tariru" and "kariru"; and another 11% were due to missing or inserted "r" phonemes, such as "sureru" versus "sueru". The systematicity of these errors leads us to believe that with more research, recognition could have been further improved by better duration constraints and other enhancements.

6.3.4. Continuous Speech Recognition Experiments

We next evaluated the LPNN system on the task of continuous speech recognition. For these experiments we used the CMU Conference Registration database, consisting of 200 English sentences using a vocabulary of 400 words, comprising 12 dialogs in the domain of conference registration, as described in Section 5.2.

In these experiments we used 40 context-independent phoneme models (including one for silence), each of which had the topology shown in Figure 6.4. In this topology, similar to the one used in the SPICOS system (Ney & Noll 1988), a phoneme model consists of 6 states, economically implemented by 3 networks covering 2 states each, with self-loops and a certain amount of state-skipping allowed. This arrangement of states and transitions provides a tight temporal framework for stationary and temporally well structured phones, as well as sufficient flexibility for highly variable phones. Because the average duration of a phoneme is about 6 frames, we imposed transition penalties to encourage the alignment path to go straight through the 6-state model. Transition penalties were set to the following values: zero for moving to the next state, *s* for remaining in a state, and *2s* for skipping a state, where *s* was the average frame prediction error. Hence 120 neural networks were evaluated during each frame of speech. These predictors were given contextual inputs from two past-frames as well as two future frames. Each network had 12 hidden units, and used sparse connectivity, since experiments showed that accuracy was unaffected while computation could be significantly reduced. The entire LPNN system had 41,760 free parameters.

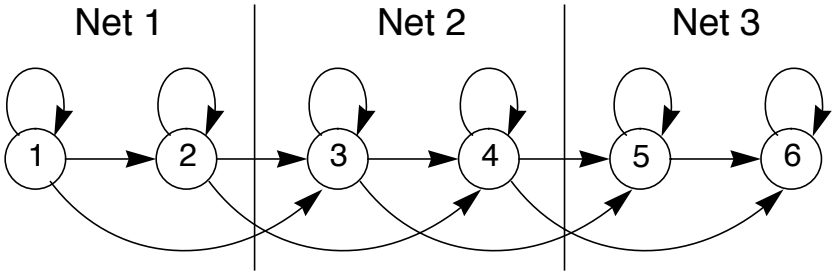


Figure 6.4: The LPNN phoneme model for continuous speech.

Since our database is not phonetically balanced, we normalized the learning rate for different networks by the relative frequency of the phonemes in the training set. During training the system was bootstrapped for one iteration using forced phoneme boundaries, and thereafter trained for 30 iterations using only "loose" word boundaries located by dithering the word boundaries obtained from an automatic labeling procedure (based on Sphinx), in order to optimize those word boundaries for the LPNN system.

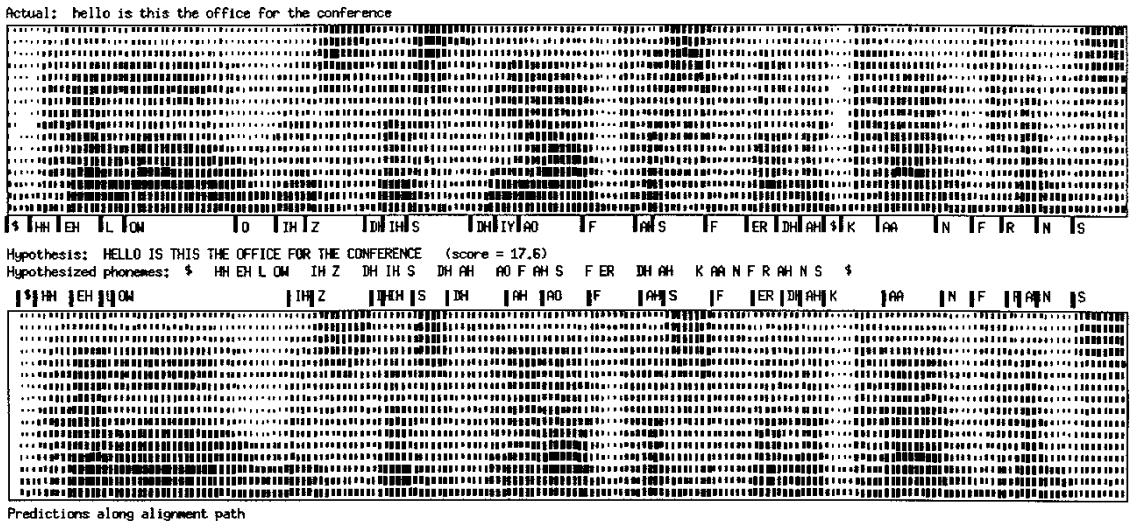


Figure 6.5: Actual and predicted spectrograms.

Figure 6.5 shows the result of testing the LPNN system on a typical sentence. The top portion is the actual spectrogram for this utterance; the bottom portion shows the frame-by-frame predictions made by the networks specified by each point along the optimal alignment path. The similarity of these two spectrograms indicates that the hypothesis forms a good acoustic model of the unknown utterance (in fact the hypothesis was correct in this case).

Speaker-dependent experiments were performed under the above conditions on two male speakers, using various task perplexities (7, 111, and 402). Results are summarized in Table 6.2.

	Speaker A			Speaker B		
Perplexity	7	111	402	7	111	402
Substitutions	1%	28%	43%	4%	28%	46%
Deletions	1%	8%	10%	2%	12%	14%
Insertions	1%	4%	6%	0%	4%	3%
Word Accuracy	97%	60%	41%	94%	56%	37%

Table 6.2: LPNN performance on continuous speech.

6.3.5. Comparison with HMMs

We compared the performance of our LPNN to several simple HMMs, to evaluate the benefit of the predictive networks. First we studied an HMM with only a single Gaussian density function per state, which we parameterized in three different ways:

$M_{16}V_0$: Mean has 16 coefficients; variance is ignored (assumed unity).

$M_{16}V_{16}$: Mean has 16 coefficients; variance has 16 coefficients.

$M_{32}V_0$: Mean has 32 coefficients (including deltas); variance is ignored.

The Gaussian means and variances in each case were derived analytically from the training data. Table 6.3 shows the results of these experiments. It can be seen that the last configuration gave the best results, but the LPNN outperformed all of these simple HMMs.

System	HMM-1 mixture			LPNN
	$M_{16}V_0$	$M_{16}V_{16}$	$M_{32}V_0$	
Substitutions	41%	35%	30%	28%
Deletions	12%	16%	13%	8%
Insertions	10%	5%	2%	4%
Word Accuracy	37%	44%	55%	60%

Table 6.3: Performance of HMMs using a single gaussian mixture, vs. LPNN.

Next we increased the number of mixture densities, from 1 to 5 to 10, where each of the Gaussians was parameterized as in $M_{32}V_0$ above, and evaluated each of these HMMs. We also compared these results against a discriminative LVQ based system developed by Otto Schmidbauer (1992), in which Learned Vector Quantization is used to automatically cluster speech frames into a set of acoustic features, which are subsequently fed into a set of neural network output units which compute the emission probability for HMM states. The results of this comparison are shown in Table 6.4. We see that an LPNN is easily outperformed by an HMM with 5 or more mixture densities, and the discriminative LVQ system outperforms everything. We attribute the inferior performance of the LPNN primarily to its lack of discrimination; this issue will be discussed in detail at the end of this chapter.

	perplexity		
System	7	111	402
HMM-1		55%	
HMM-5	96%	70%	58%
HMM-10	97%	75%	66%
LVQ	98%	80%	74%
LPNN	97%	60%	40%

Table 6.4: Word accuracy of HMM-*n* with *n* mixture densities, LVQ, and LPNN.

Finally, we measured the frame distortion rate of each of the above systems. In an LPNN, frame distortion corresponds to the prediction error. In an HMM, it corresponds to the distance between the speech vector and the mean of the closest gaussian in the mixture. In the LVQ system, it corresponds to the quantization error, i.e., the distance between the input vector and the nearest weight vector of any hidden node. Table 6.5 shows that the LPNN has the least distortion rate of any of these systems, despite its inferior word accuracy. This suggests that the training criterion, which explicitly minimizes the frame distortion rate, is inconsistent and poorly correlated with the ultimate goal of word recognition accuracy. We will further discuss the issue of consistency in the next chapter (Section 7.4).

System	Avg. Frame Distortion
HMM-1	0.15
HMM-5	0.10
HMM-10	0.09
LVQ	0.11
LPNN	0.07

Table 6.5: The LPNN has minimal frame distortion, despite its inferior word accuracy.

6.4. Extensions

In our attempts to improve the accuracy of our LPNN system, we investigated several extensions to the basic system. This section describes those architectural extensions, and presents the results of those experiments.

6.4.1. Hidden Control Neural Network

A common theme in speech recognition systems is to balance the number of free parameters against the amount of training data available, in order to optimize accuracy on the test set. If there are too many free parameters, the system may learn to perfectly memorize the training set, but will generalize poorly to new data. On the other hand, if there are too few free parameters, the system will learn only the coarse characteristics of the task, and so will attain poor accuracy on both the training set and the testing set. Striking an optimal balance always involves sharing parameters to some extent. In a pure HMM system, this can mean sharing codebooks across all states, sharing distributions across states within a phoneme, merging triphones into generalized triphones, merging distributions via senones, and so on. In a NN-HMM hybrid, many of these techniques can still be used; for example, in the last section we described a 6-state phoneme model that uses only 3 networks, sharing distributions across states.

Another way to share parameters, which is unique to neural networks, is to collapse multiple networks into a single network, modulated by a "hidden control" input signal that distinguishes between the functionality of the separate networks. This idea was initially proposed

by Levin (1990), and called a *Hidden Control Neural Network* (HCNN). In the context of speech recognition, this involves collapsing multiple predictive networks into a shared HCNN network, modulated by a hidden control signal that distinguishes between the states, as shown in Figure 6.6. The control signal typically uses a simple *thermometer representation*, comprised of one unit for each state, where successive states are represented by turning on successive units, ensuring a similar representation for adjacent states. In addition to reducing the number of parameters (and thus the amount of memory required), the HCNN can also be computationally more efficient than a set of separate networks, since partial results of redundant forward passes can be cached (although the total number of forward passes remains unchanged).

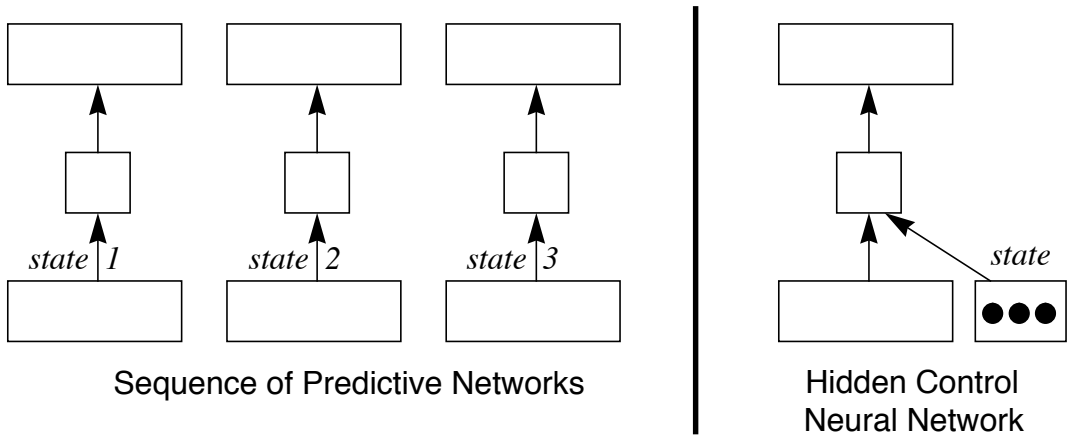


Figure 6.6: A sequence of predictive networks can be replaced by a Hidden Control Neural Network.

We performed a number of experiments with the HCNN, in collaboration with Bojan Petek (1991, 1992). In one set of experiments, we studied the effects of varying degrees of shared structure, as shown in Figure 6.7. These experiments used 2-state phoneme models, rather than 3- or 6-state phoneme models. The first architecture, labeled (a), was a basic LPNN (i.e., no hidden control), in which 80 networks are required to represent 40 phonemes with 2 states each. In (b), hidden control inputs were introduced such that only 40 networks are required for the same task, as each phoneme is modeled by a single network modulated by 2 hidden control input bits which distinguish between the two states. In (c), the hidden control idea is taken to its limit: one big network is modulated by 40 x 2 = 80 hidden control inputs which specify both the phoneme and the state.

Table 6.6 shows the results of these experiments, evaluated on speaker B. Besides testing continuous speech recognition, we also tested excerpted word recognition, in which word boundaries within continuous speech are given; this allowed us to compare the acoustic discriminability of the three architectures more directly. As the table shows, we observed minor differences in performance between architectures (a) and (b): the LPNN was slightly more discriminant, but the hidden control architecture generalized better and ran faster. Meanwhile, architecture (c) did very poorly, presumably because it had too much shared structure and too few free parameters, overloading the network and causing poor discrimi-

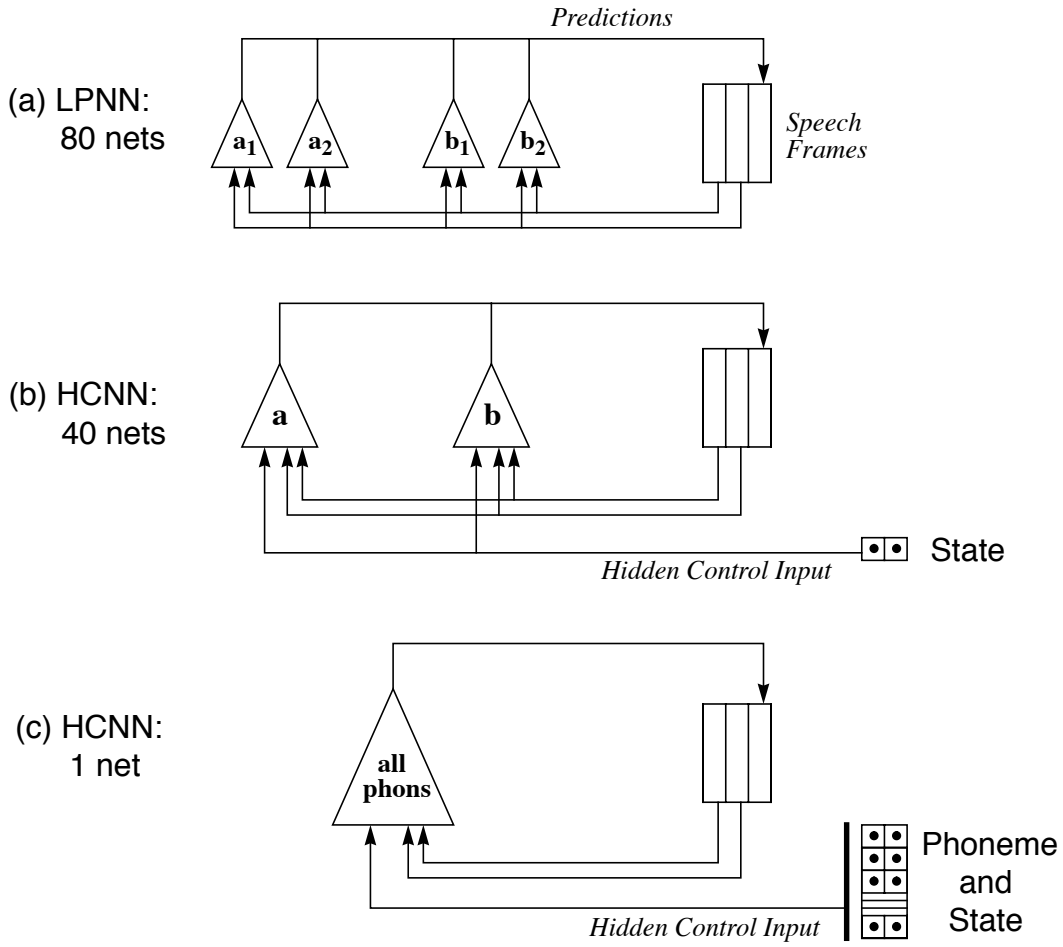


Figure 6.7: Architectures used in Hidden Control experiments.

nation. Hence, we conclude that hidden control may be useful, but care must be taken to find the optimal amount of parameter sharing for a given task.

Architecture	(a)	(b)	(c)
# free parameters (weights)	80960	42080	6466
Word accuracy:			
Excerpted words (P=402)	70%	67%	39%
Continuous speech (P=7)	91%	91%	n/a
Continuous speech (P=402)	14%	20%	n/a

Table 6.6: Results of Hidden Control experiments. Parameter sharing must be used sparingly.

6.4.2. Context Dependent Phoneme Models

The accuracy of any speech recognizer can be improved by using context dependent models. In a pure HMM system, this normally involves using diphone or triphone models, i.e., a phoneme model in the context of one or both of its adjacent phonemes. This increases the specificity and accuracy of the models, but also increases their absolute number by orders of magnitude (e.g., from 40 to 1600 to 64000 models), such that it becomes necessary to cluster them (another form of parameter sharing), to ensure that there is still enough training data per model.

In a NN-HMM hybrid based on predictive networks, context dependent phoneme models could be implemented by using a separate network for each diphone or triphone. However, this would undoubtedly result in a system with too many free parameters, resulting in poor generalization (and an excessive amount of memory and computation). What is desired is again some form of parameter sharing. One potential solution is to use a shared network in which the context is a part of the input signal, as in the HCNN. This approach is appealing because it requires very few additional parameters (diphones require only c extra inputs, and triphones require only $2c$ extra inputs, for some small value of c), and yet it provides a way to distinguish between all the different contexts of a phoneme.

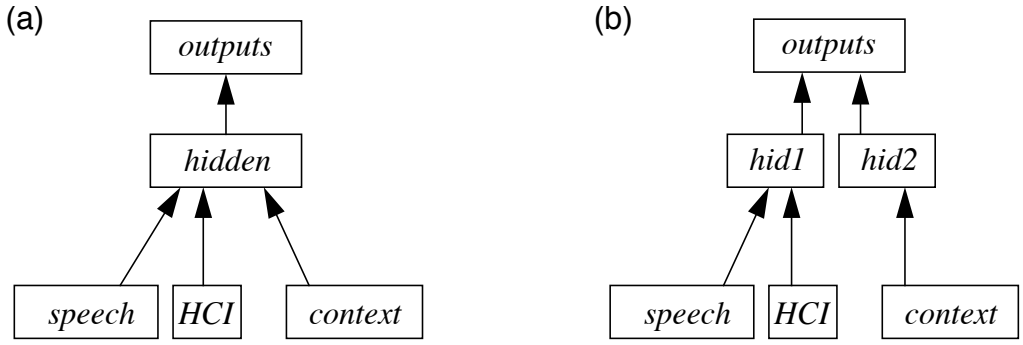


Figure 6.8: Context-dependent HCNN: (a) standard implementation; (b) efficient implementation.

We studied this idea by augmenting our HCNN network to include contextual inputs, as shown in Figure 6.8(a). Our contextual inputs represented only one adjacent phoneme, making this a right-context dependent diphone model (we felt that our database was too small to provide adequate coverage of triphones). We could have represented the 40 possible values of the adjacent phoneme using 40 contextual inputs, but instead we clustered the phonemes by their linguistic features, as proposed by (Rumelhart & McClelland 1986: chapter 18), so that only 10 contextual inputs were necessary. Each phoneme was coded along four dimensions. The first dimension (three bits) was used to divide the phonemes into interrupted consonants (stops and nasals), continuous consonants (fricatives, liquids, and semivowels), and vowels. The second dimension (two bits) was used to subdivide these classes. The third dimension (three bits) classified the phonemes by place of articulation

(front, middle, back). Finally, the fourth dimension (two bits) divided the consonants into voiced and unvoiced, and vowels into long and short.

Conceptually there is only a single hidden layer in the predictive network. But in reality, we divided this hidden layer into two halves, as shown in Figure 6.8(b). This allows the forward pass computations on each half of the network to be cached, so that for a given frame and state, the forward pass computations over all contexts can be reduced to a series of output sigmoids using different precomputed net inputs. This saves a considerable amount of redundant computation.

We evaluated the context-dependent HCNN on the CMU Conference Registration database. Our best results are shown in Table 6.7 (for speaker A, perplexity 111). In this evaluation, the predictive network's inputs included 64 speech inputs (2 frames of speech represented by 16 melscale coefficients and 16 delta coefficients), 5 state inputs, and 10 contextual inputs; the network also included 30 hidden units on the speech side, plus 5 hidden units on the context side; and of course 16 outputs representing the predicted speech frame. As in the LPNN, all phonemes used two alternate models, with the best one automatically linked in. In contrast to the LPNN, however, which used a 6-state phoneme model implemented by 3 networks, this context-dependent HCNN used the 5-state phoneme model shown in Figure 6.9 (or a 3-state model for silence), implemented by a single network per phoneme with state inputs. The CD-HCNN achieved much better results than the LPNN (72% vs. 60%), suggesting that the hidden control mechanism provides an effective way to share parameters, and that context dependence improves the specificity of its phoneme models.

System	LPNN	CD-HCNN
Substitutions	28%	20%
Deletions	8%	6%
Insertions	4%	2%
Word accuracy	60%	72%

Table 6.7: Results of LPNN and context-dependent HCNN (speaker A, perplexity 111).

An error analysis at the phoneme level revealed that there was still a phoneme error rate of 20% after training was complete. Most of the confusions involved the phonemes `AH,AE, EH,UH,D,K,N`.

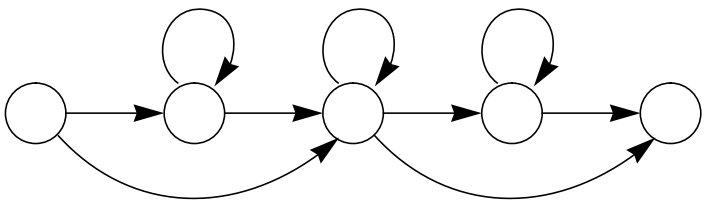


Figure 6.9: 5-state phoneme model used in CD-HCNN experiments.

6.4.3. Function Word Models

Lee (1988) showed that function words — short words like "a", "of", "that" — are particularly difficult to recognize, because they have strong coarticulation effects, are very frequent in continuous speech, and are often poorly articulated by the speaker. Inadequate modeling of these words can significantly degrade a system's overall word accuracy.

We improved the accuracy of our system by introducing function word models. We selected the three words with the highest error rate in our system ("a", "the", "you"), and created whole-word models for these, with states indexed by a hidden control inputs. That is, rather than representing these words by a sequence of standard phoneme models, these words got independent models, each of which was represented by a single HCNN with from two to five additional inputs to identify the state of the word (as in Levin 1990). These function word models were also context-dependent; the contextual inputs were arbitrarily set to the initial phoneme of the function word. Note that because of the mutual independence of the predictive networks, there was no need to retrain the other phoneme models when these new function word models were trained.

Evaluating this system under the same conditions as in the previous section, we found that this system achieved 75% word accuracy, which represents 10% fewer errors than the system without function word models.

6.5. Weaknesses of Predictive Networks

Our experience suggests that predictive networks are not very effective for speech recognition. On the Conference Registration database at perplexity 111, we obtained only 60% word accuracy with our basic LPNN, or 75% when the system was enhanced by hidden control inputs, context dependent modeling, and function word modeling. By contrast, a primitive HMM also achieves 75% on this task, and a simple LVQ based system achieves 80% word accuracy.

We have concluded that predictive networks suffer from two weaknesses: (1) a lack of discrimination, and (2) inconsistency between training and testing criteria. It may be possible to correct these problems, but our research stopped short of doing so. We discuss these problems in the following sections.

6.5.1. Lack of Discrimination

Predictive networks are ordinarily trained independently of each other; as a result, there is no discrimination between the acoustic models. This means there is no explicit mechanism to discourage models from resembling each other, which leads to easily confusable phone models, which in turn degrades word recognition accuracy. This weakness is shared by HMMs that are trained with the Maximum Likelihood criterion; but the problem is more severe for predictive networks, because the quasi-stationary nature of speech causes all of

the predictors to learn to make a quasi-identity mapping, rendering all of the phoneme models fairly confusable. For example, Figure 6.10 shows an actual spectrogram and the frame-by-frame predictions of the /eh/ model and the /z/ model. Disappointingly, both models are fairly accurate predictors for the entire utterance.

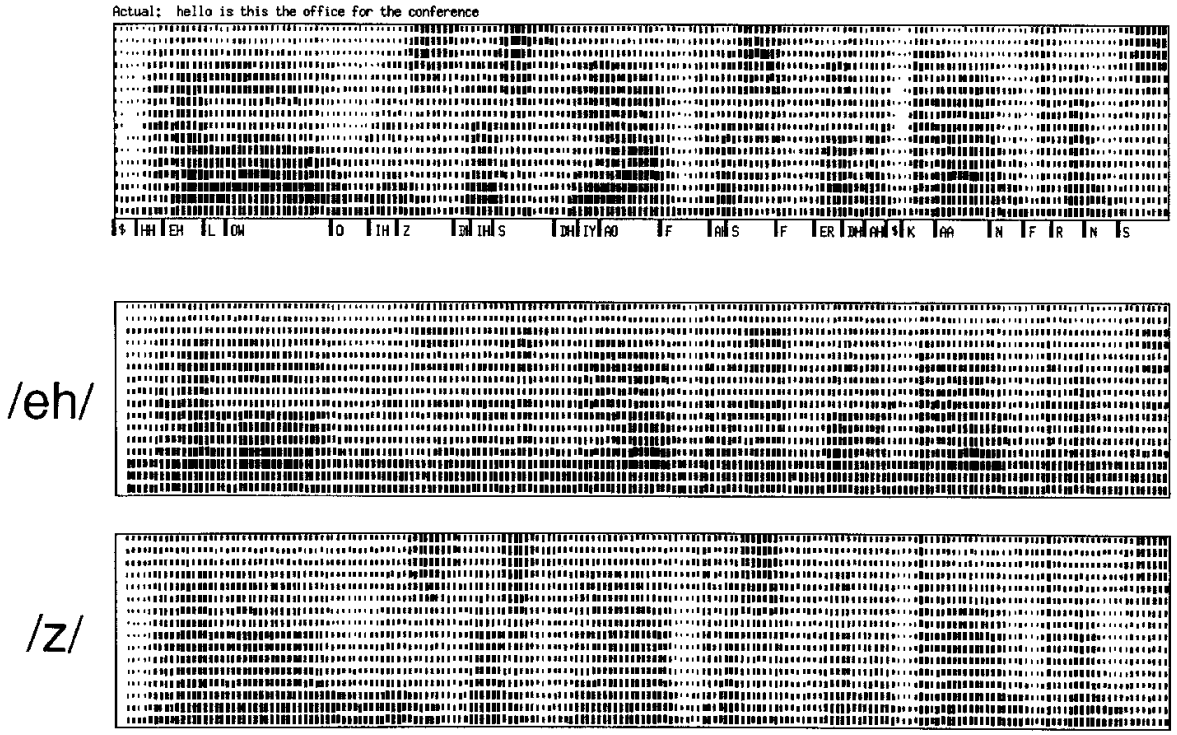


Figure 6.10: Actual spectrogram, and corresponding predictions by the /eh/ and /z/ phoneme models.

There are many ways to partially compensate for this lack of discrimination. For example, we can use more input frames (as long as they are temporally close enough to be relevant to the predicted frame), thereby making each network behave less like an identity mapper, so they can be more easily distinguished. Or we can introduce alternate phone models, or context dependent models, or function word models, or a number of other improvements. However, while each of these techniques may improve the performance of the system, they do not address the lack of discrimination between models, so performance will always remain suboptimal.

What is really needed is a way to discriminate between models, by applying *positive training* to the correct model while somehow applying *negative training* to all of the incorrect models. However, it is not immediately clear what kind of negative target makes sense for a predictive network. In hindsight, we see that there are two general types of target that could be used during both positive and negative training: (1) a vector, analogous to the predicted frame; or (2) a scalar, corresponding to some transformation of the predicted frame (or of all predicted frames). In our research, we studied the first approach, but we had not yet thought of the second approach; it now appears that the second approach may have more promise. The remainder of this section will describe these two approaches.

6.5.1.1. Vector targets

If a prediction-based system uses vectors as training targets, then by definition the target for positive training is the actual frame at time *t*, but there is no obvious target vector for negative training. In our attempts to perform negative training, we studied two possible strategies, neither of which was successful.

The first strategy was to use the actual frame at time *t* as a target for both positive and negative training, but to perform gradient *descent* for the correct network and gradient *ascent* for all incorrect networks. This rather naive approach failed because the force of discrimination is proportional to the prediction error of the network, such that negative training is weakest for the most confusable model, and becomes stronger for all models that have already been pushed away. This is an unstable dynamic, which inevitably throws the models into chaos.

The second strategy returned to using gradient descent for both positive and negative training, but tried to supply a target vector for negative training that would distinguish each model from the others. We observed that each network receives positive training in only a small region of acoustic space (i.e., those frames corresponding to that phoneme), and consequently for any other input frames it will compute an undefined output, which may overlap with the outputs of other predictors. If the network has learned to approximate an identity mapping in its defined region, then it will also tend to approximate an identity mapping in its undefined region. To discourage this behavior, we applied negative training throughout this undefined region, using a target that differed from the actual frame at time *t*. We chose the negative target vector to be the average of all the positive frames associated with that network; this clearly avoids identity mappings, because, for example, the `A` model is trained to predict an average `A` frame whenever the input frames belong to `B`. Unfortunately, this technique failed because each network learned to compute an essentially constant output, corresponding to the average frame of its associated phone. This happened, naturally, because the network was trained to map virtually any input to a constant output, except for a few positive predictions, which also happened to resemble that constant output. We tried to sensitize our networks by using a smaller learning rate for negative training (to emphasize the positive samples) and by increasing the size of our networks (so they could learn more detailed mappings); but our efforts were not successful.

From our experience we have concluded that discriminative training in a predictive system is at best nontrivial and probably infeasible using vector targets.

6.5.1.2. Scalar Targets

An interesting alternative involves redefining the boundaries of a predictive network, so that the associated prediction error (i.e., the Euclidean distance between the predicted frame and the actual frame) is computed internally to the network by a special post-processing layer. With this perspective, the output of the network is a scalar, equal to the Euclidean distance

$$d = \sum_{i}^{N} (y_i - t_i)^2 \tag{65}$$

which lies in the range [0..*N*] if each frame has *N* coefficients in the range [0..1]. This can be transformed to a class membership function by inverting and normalizing it, using

$$z = \exp(-d) \tag{66}$$

so that a perfect prediction yields $z = 1$, and a poor prediction yields $z \approx 0$. Now discriminative training is simply a matter of training the correct network on the target $T = 1$, and all of the incorrect networks on the target $T = 0$. Of course, error must now be backpropagated through the equations in the post-processing layer. If we assume the squared error criterion

$$E = \frac{1}{2}(z - T)^2 \tag{67}$$

for each network, then at the frame prediction layer we have

$$\begin{aligned} \frac{\partial E}{\partial y_i} &= \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial d} \cdot \frac{\partial d}{\partial y_i} \\ &= (z - T) \cdot (-z) \cdot (2(y_i - t_i)) \end{aligned} \tag{68}$$

Note that this backpropagation equation involves two types of targets: *T* at the class membership layer, and t_i at the frame prediction layer. It can be seen that this learning rule causes the system to discriminate between models because $(z - T)$ is negative for the correct network and positive for all incorrect networks.

A variation of this approach is to first normalize the scalar outputs so that they sum to 1, like probabilities:

$$p^j = \frac{z^j}{\sum_k z^k} \qquad \text{so} \qquad \sum_j p^j = 1 \tag{69}$$

where superscripts indicate the network index. As before, the correct network is trained on the target $T = 1$, and all the incorrect networks are trained on the target $T = 0$. This time, if the error measure is still

$$E = \frac{1}{2}\sum_j (p^j - T^j)^2 \tag{70}$$

then at the frame prediction layer we have

$$\begin{aligned}\frac{\partial E}{\partial y_i^k} &= \sum_j \frac{\partial E}{\partial p^j} \cdot \frac{\partial p^j}{\partial z^k} \cdot \frac{\partial z^k}{\partial d^k} \cdot \frac{\partial d^k}{\partial y_i^k} \\ &= \sum_j \left[p^j - T^j\right] \cdot \left[\frac{\delta_{jk} \sum_k z^k - z^j}{\left(\sum_k z^k\right)^2}\right] \cdot \left[-z^k\right] \cdot \left[2\left(y_i^k - t_i\right)\right] \\ &= \sum_j \left[p^j - T^j\right] \cdot \left[p^k\left(p^j - \delta_{jk}\right)\right] \cdot \left[2\left(y_i^k - t_i\right)\right]\end{aligned} \tag{71}$$

Mellouk and Gallinari (1993) used such normalized scalar targets to introduce discrimination into a predictive system that resembled our basic LPNN. Although their system was a continuous speech recognizer, they evaluated its performance on phoneme recognition. In their preliminary experiments they found that discriminative training cut their error rate by 30%. A subsequent test on the TIMIT database showed that their phoneme recognition rate of 68.6% was comparable to that of other state-of-the-art systems, including Sphinx-II.

Normalized outputs are somewhat more discriminative than non-normalized outputs, since normalized outputs are mutually constrained so that when the correct one increases, all of the incorrect ones will decrease. This property might be called *implicit discrimination*. By contrast, *explicit discrimination* involves contrasting positive and negative training, i.e., training the correct model to output a 1 while training all incorrect models to output a 0. Note that these two modes of discrimination operate independently of each other. Explicit discrimination probably has more impact than implicit discrimination, since it involves greater contrast. Nevertheless, it may be best to combine both types of discrimination, as Mellouk and Gallinari have done.

We conclude that a predictive system *can* become discriminative by transforming the vector outputs to scalar outputs, so that the network can be trained on targets of 0 and 1. However, we did not empirically investigate this approach in our research.

6.5.2. Inconsistency

The second major problem with predictive networks is that their standard training criterion is inconsistent with the testing criterion. That is, predictive networks are trained to make accurate predictions of speech frames, but the testing criterion is something completely different, i.e., word recognition accuracy. We hope against hope that these two criteria are strongly correlated, but in fact we find that the LPNN's excellent frame predictions translate to poor word recognition accuracy. Evidently there is only a weak correlation between frame prediction and word recognition.

Training and testing could be made more consistent by extending the architecture to support *word level training*. This would involve introducing a word level unit for each word in the vocabulary, connecting it to the prediction error layer along its associated DTW alignment path, and backpropagating error down from the word layer using a target of 1 for the

correct word and 0 for incorrect words. We will discuss the technique of word level training in greater detail in the next chapter, in the context of classification networks.

In conclusion, predictive networks suffer from two major weaknesses, i.e., a lack of discrimination, and inconsistency between the training and testing criteria. We have discussed some potential remedies for each of these problems. Rather than actually pursuing these remedies in our research, however, we chose to move on and study classification networks, because they support discrimination much more naturally, and they appeared likely to give superior results. Our research with classification networks is described in the next chapter.

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?

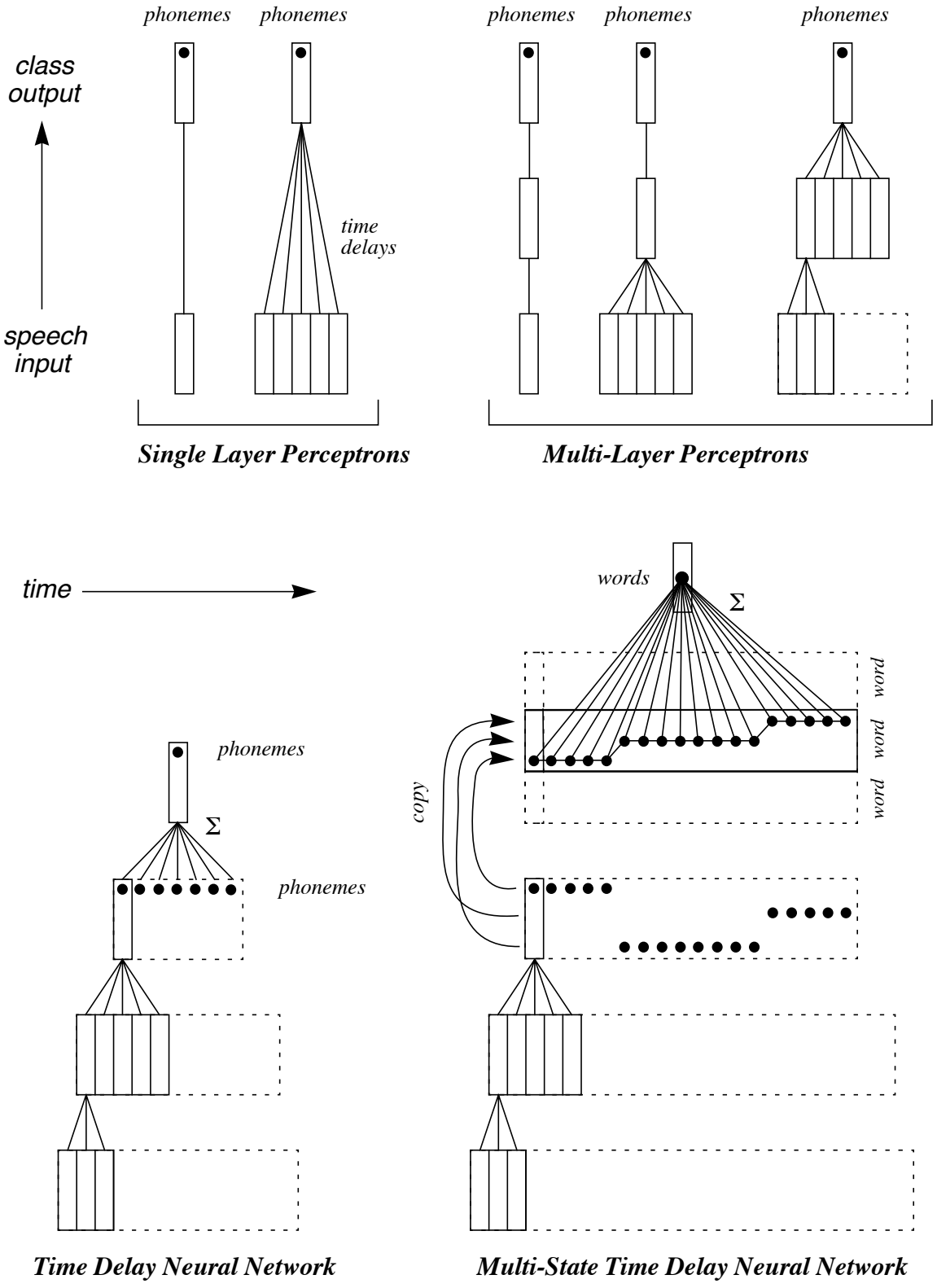


Figure 7.1: Types of network architectures for classification.

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.

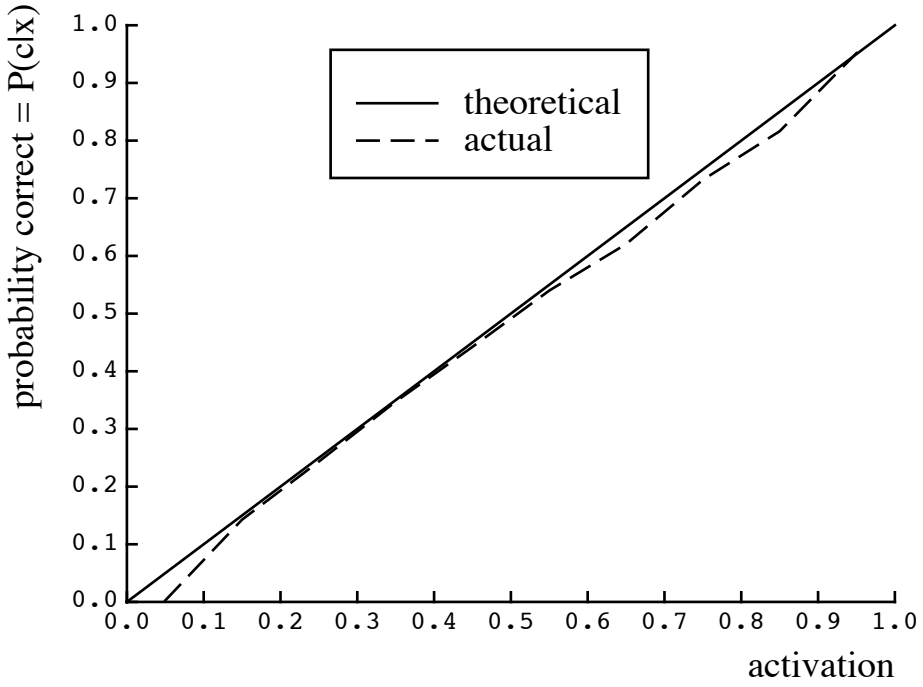


Figure 7.2: Network output activations are reliable estimates of posterior class probabilities.

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 $\sum_i P(c_i|x) = 1$.

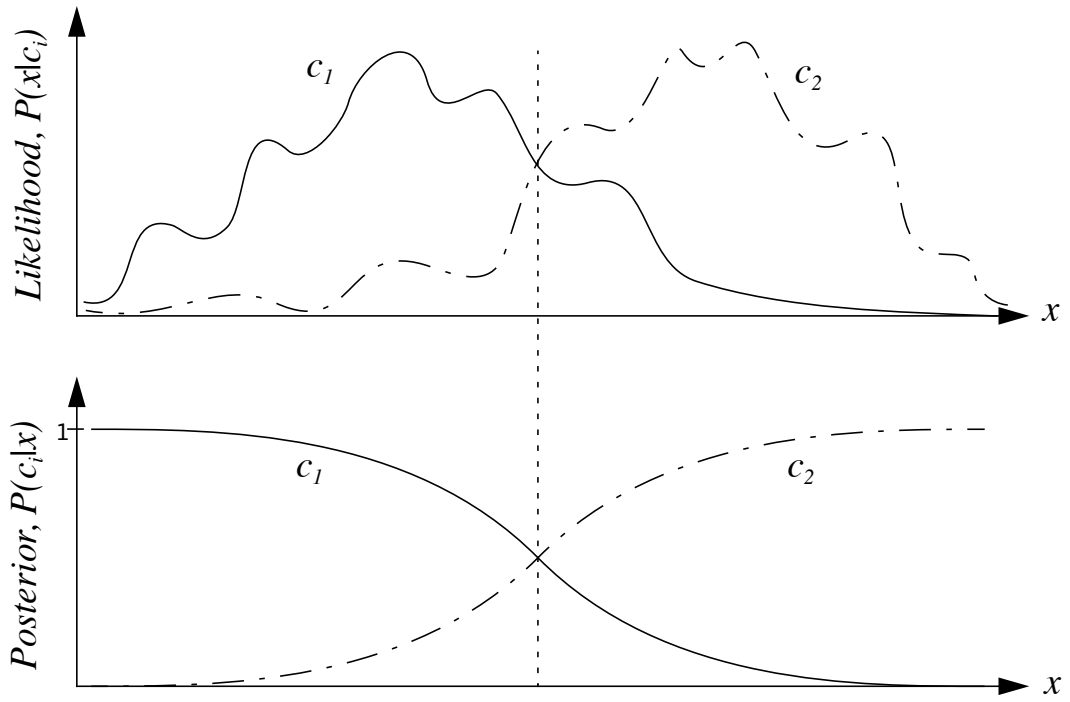


Figure 7.3: Likelihoods model independent densities; posteriors model their comparative probability.

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 ⇒ 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.

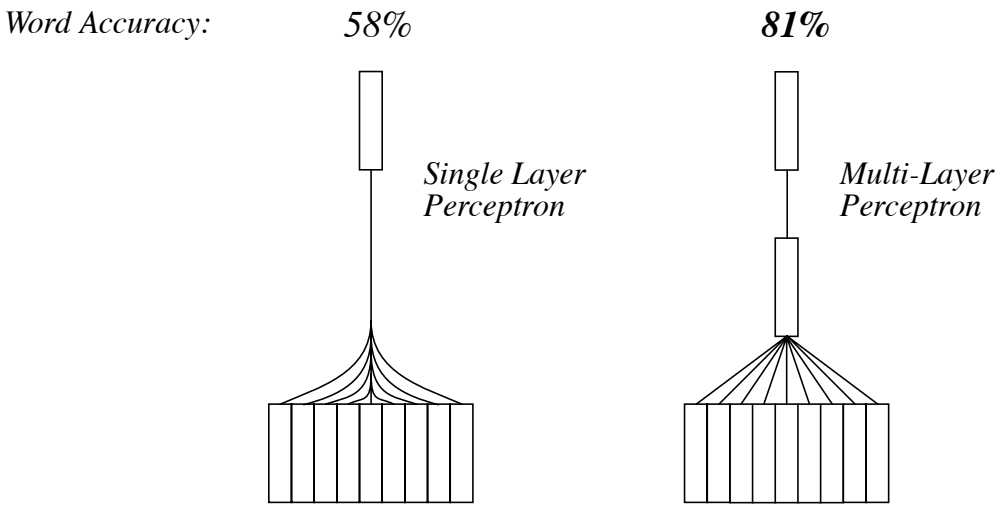


Figure 7.4: A hidden layer is necessary for good word accuracy.

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.

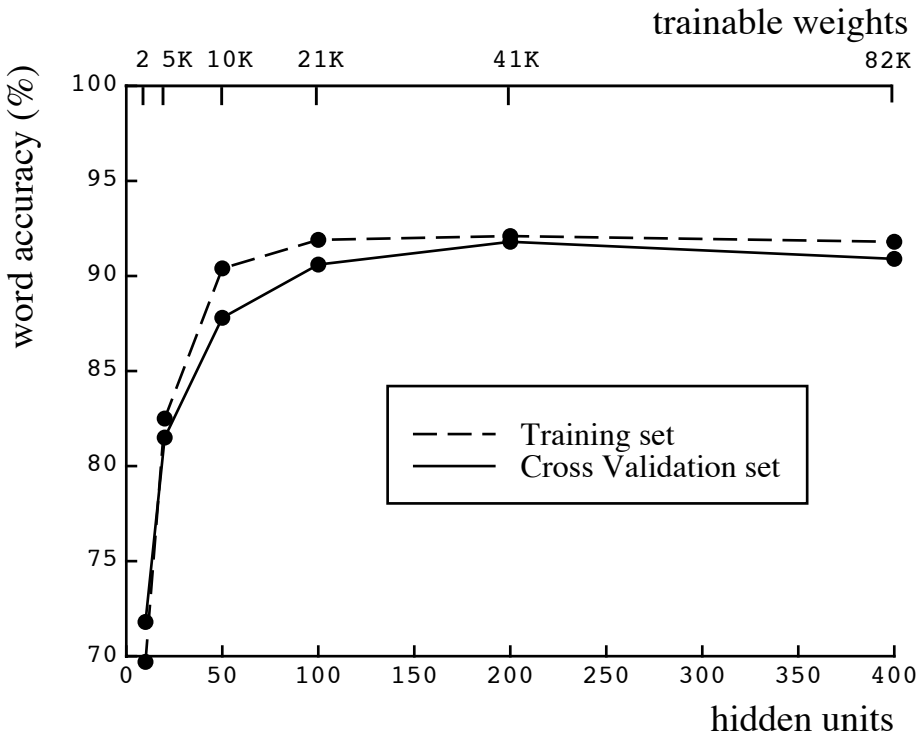


Figure 7.5: Performance improves with the number of hidden units.

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.

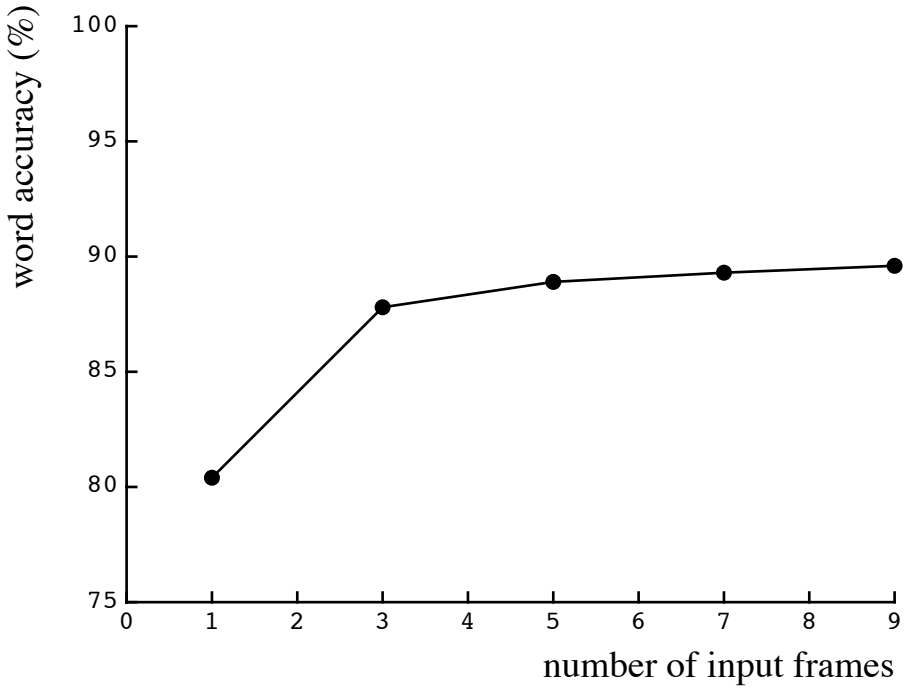


Figure 7.6: Enlarging the input window enhances context sensitivity, and so improves word accuracy.

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-

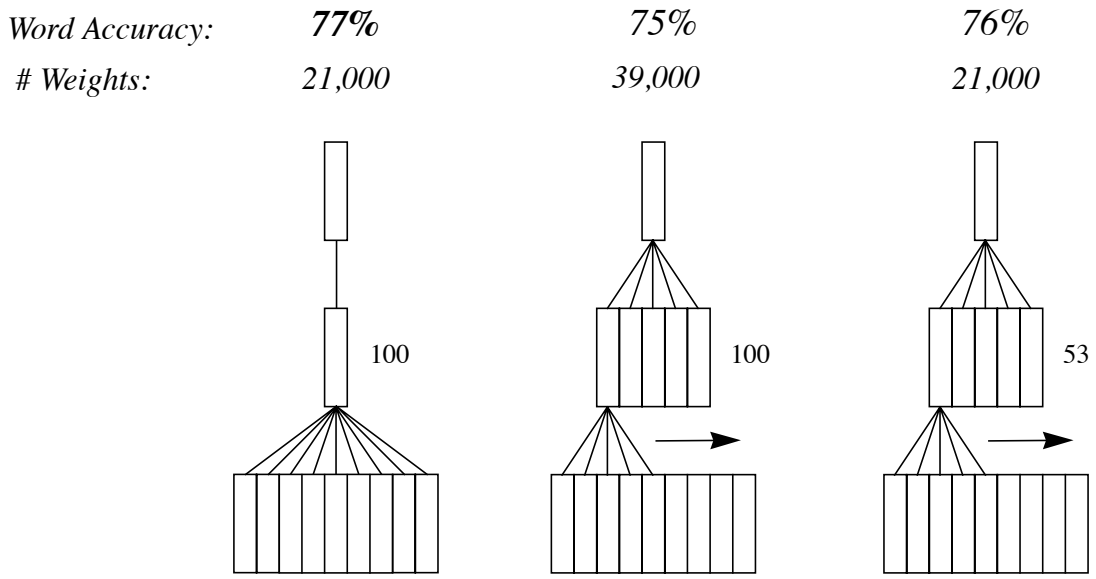


Figure 7.7: Hierarchical time delays do not improve performance when there is abundant training data.

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-

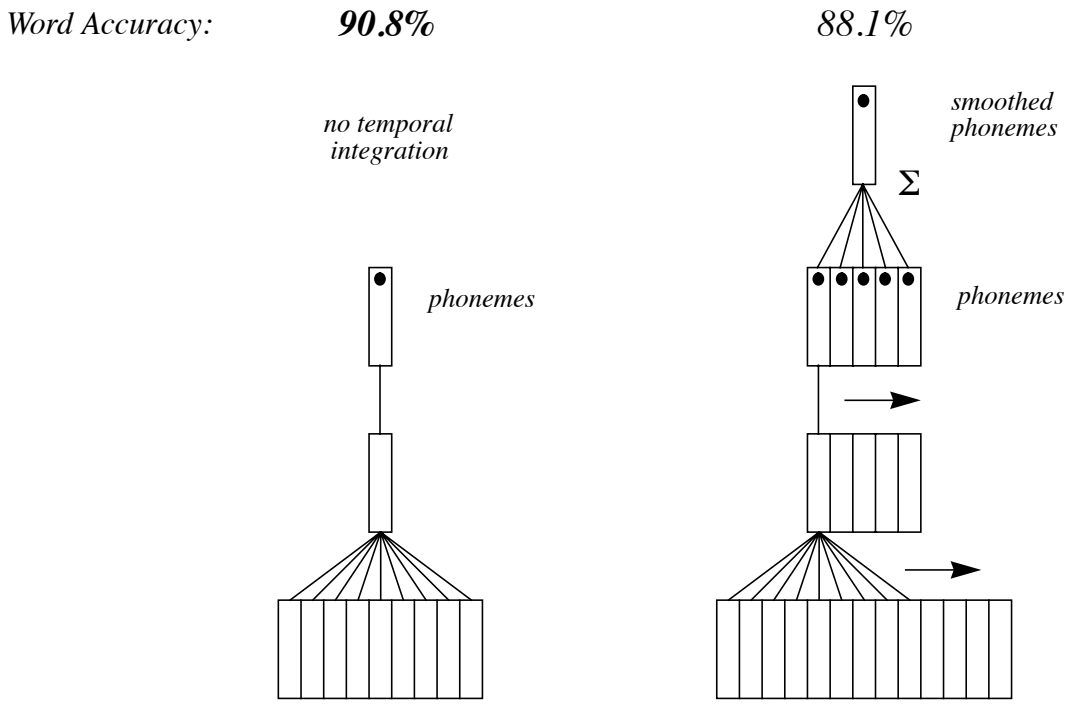


Figure 7.8: Temporal integration of phoneme outputs is redundant and not helpful.

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

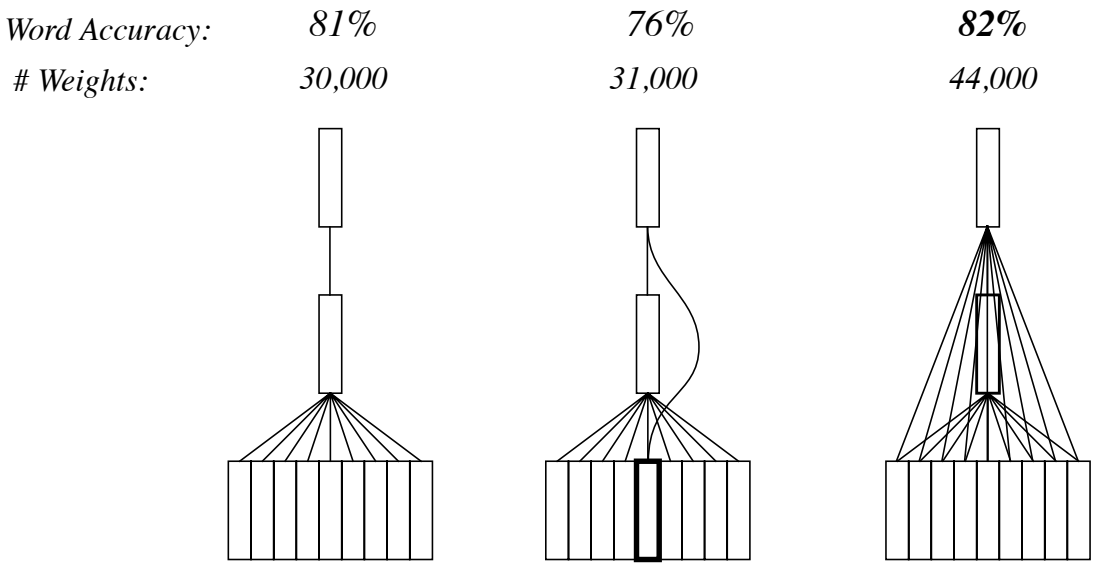


Figure 7.9: Shortcut connections have an insignificant advantage, at best.

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.

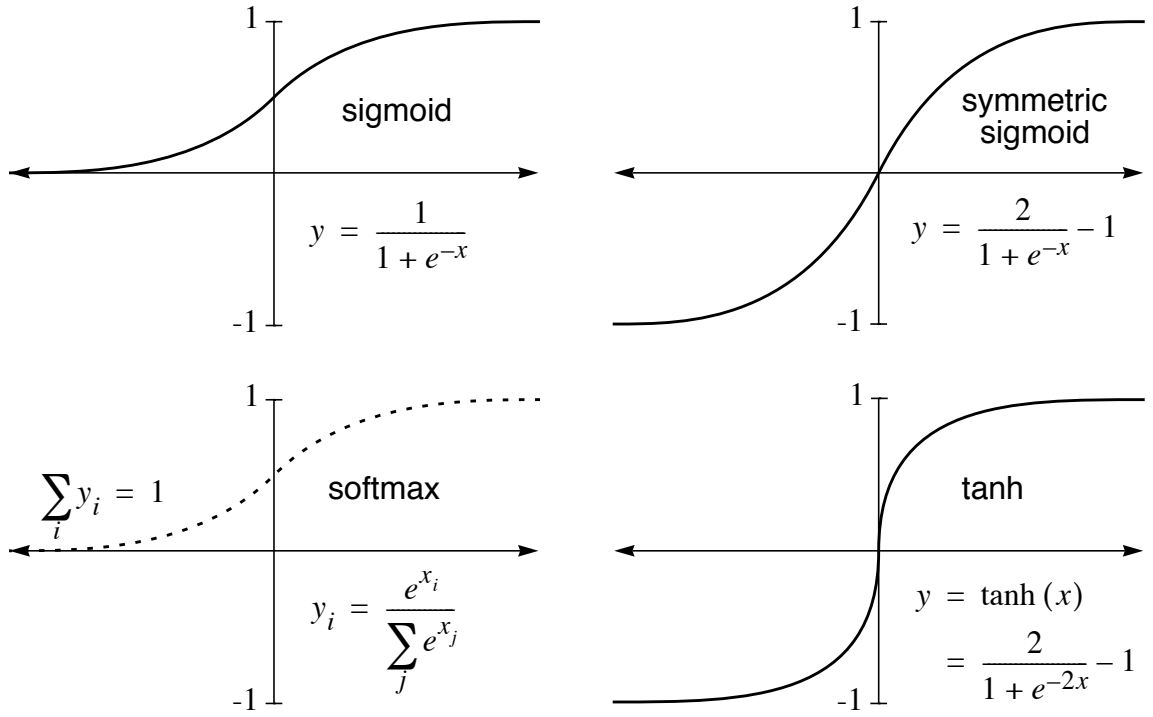


Figure 7.10: Four popular transfer functions, for converting a unit's net input *x* to an activation *y*.

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.

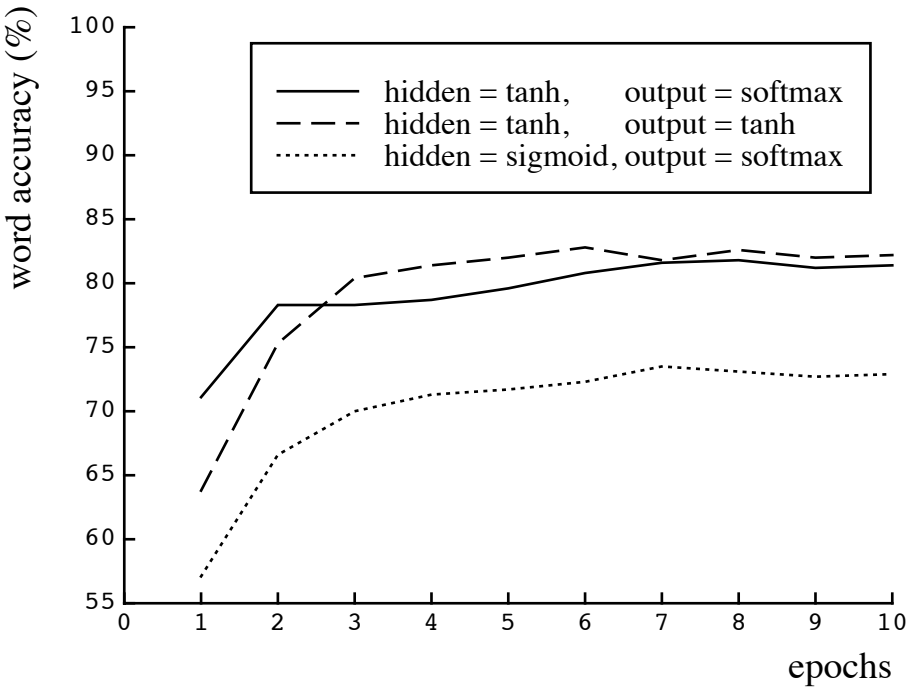


Figure 7.11: Results of training with different transfer functions in the hidden and output layers.

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

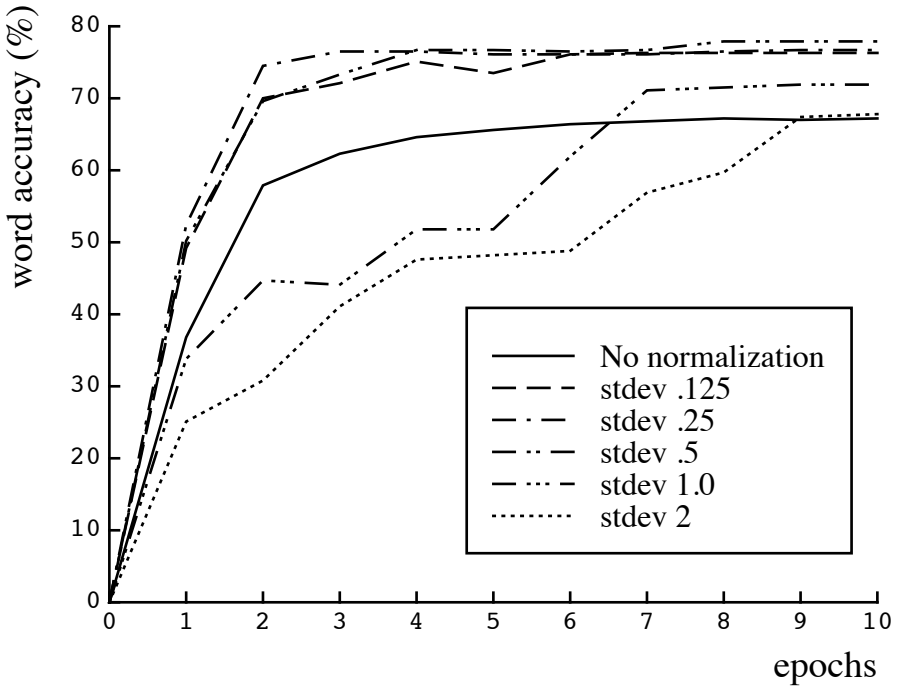


Figure 7.12: Normalization of PLP inputs is very helpful.

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.

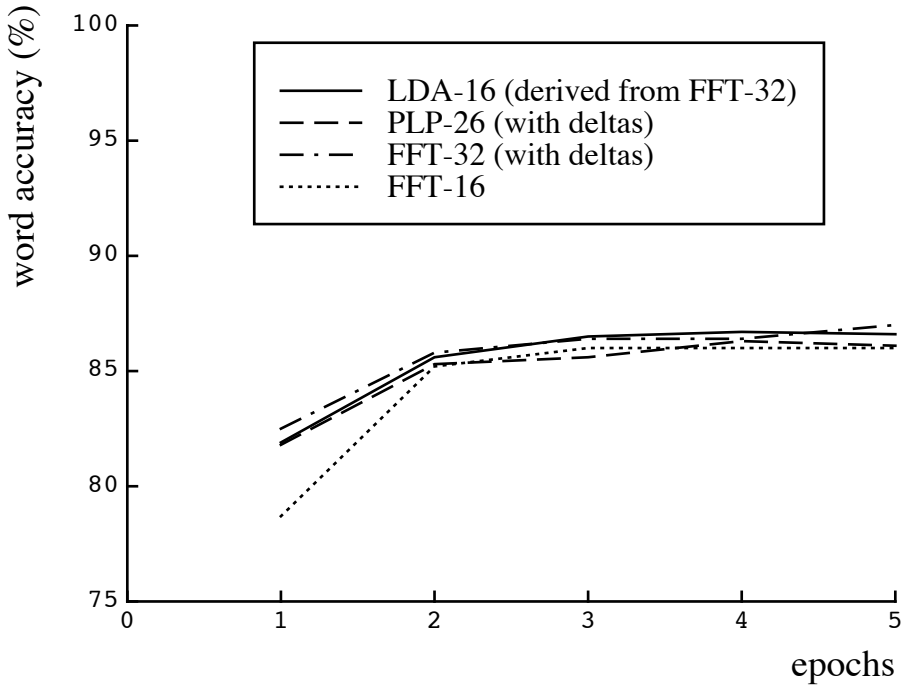


Figure 7.13: Input representations, all normalized to [-1..1]: Deltas and LDA are moderately useful.

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.

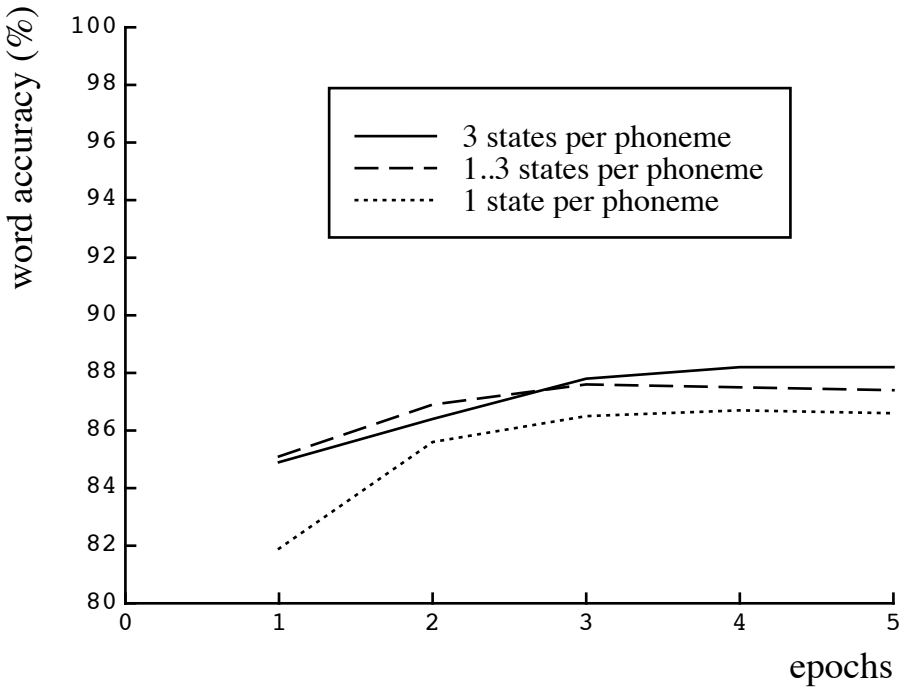


Figure 7.14: A 3-state phoneme model outperforms a 1-state phoneme model.

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.

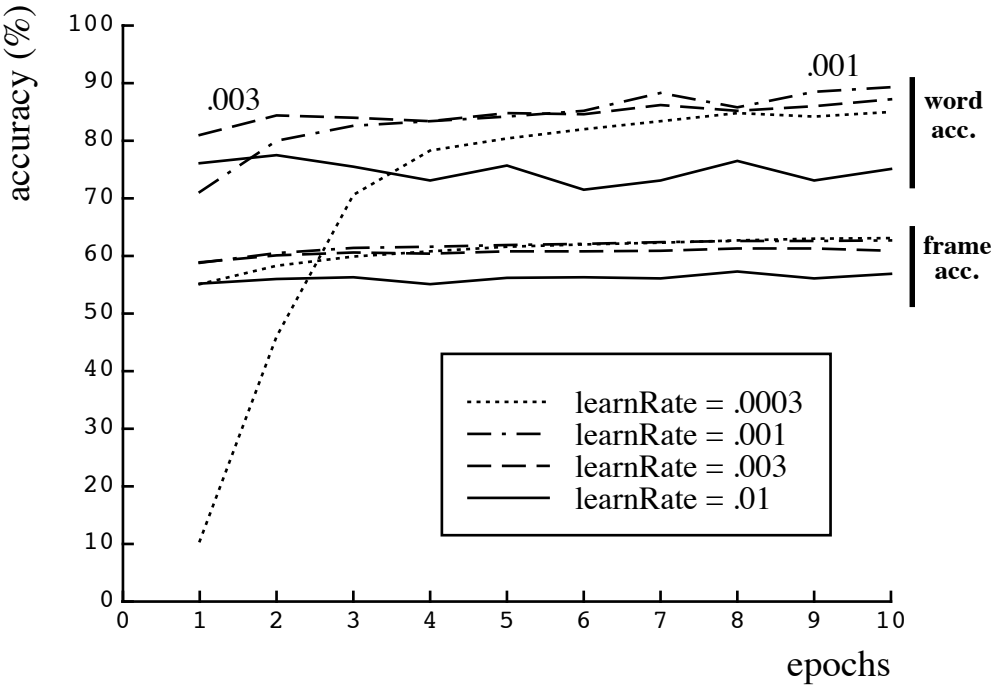


Figure 7.15: Constant learning rates are unsatisfactory; the learning rate should decrease over time.

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*

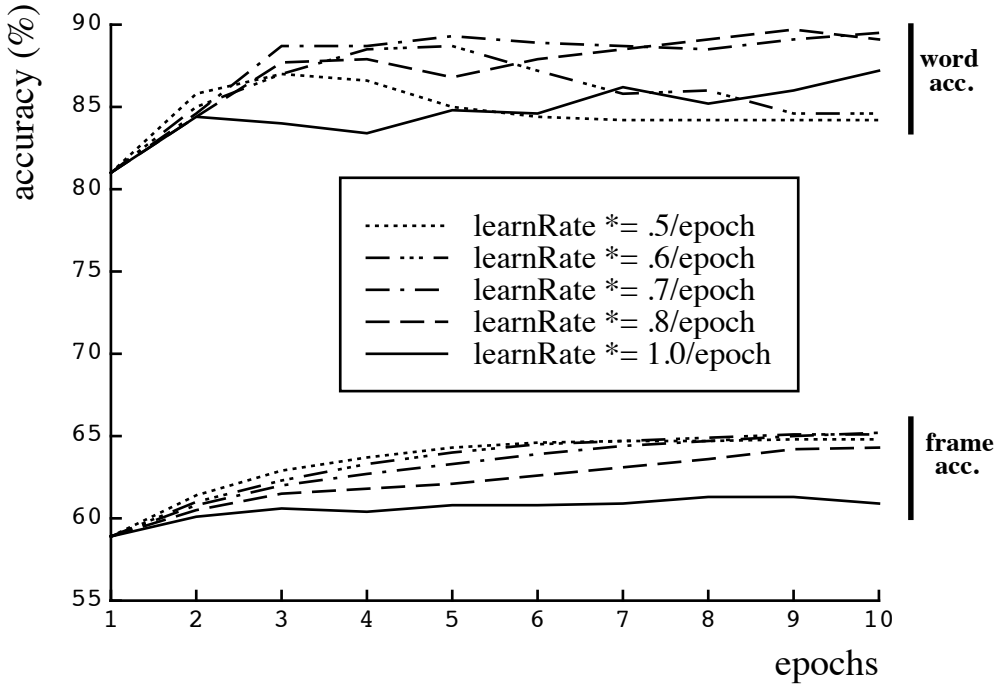


Figure 7.16: Geometric learning rates (all starting at LR = .003) are better, but still may be suboptimal.

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.

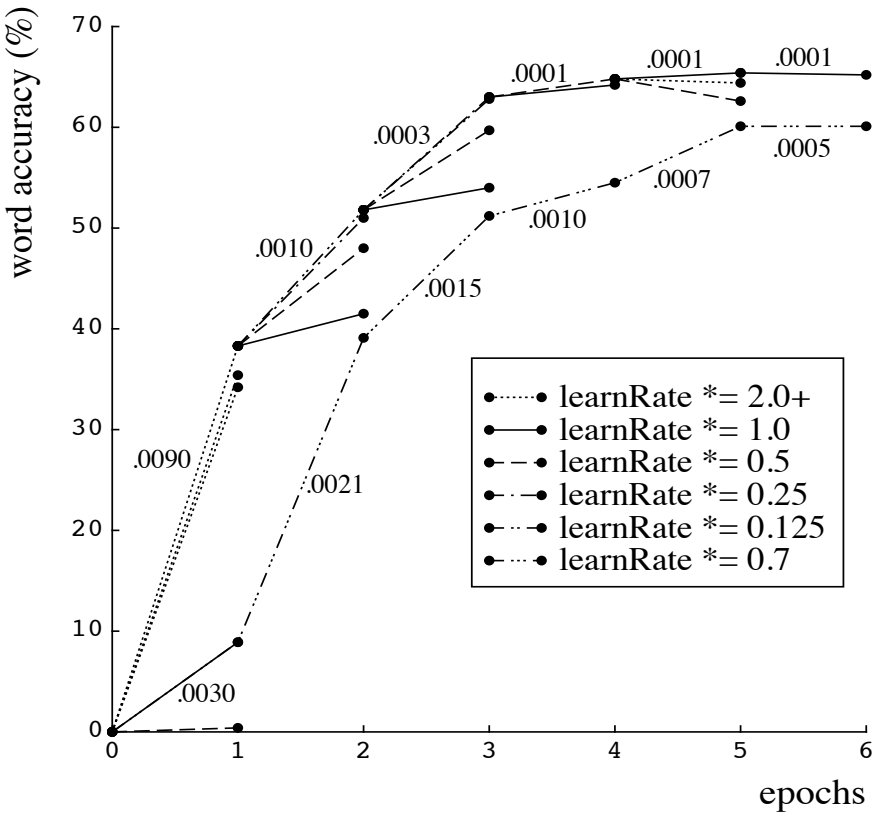


Figure 7.17: Searching for the optimal learning rate schedule.

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.

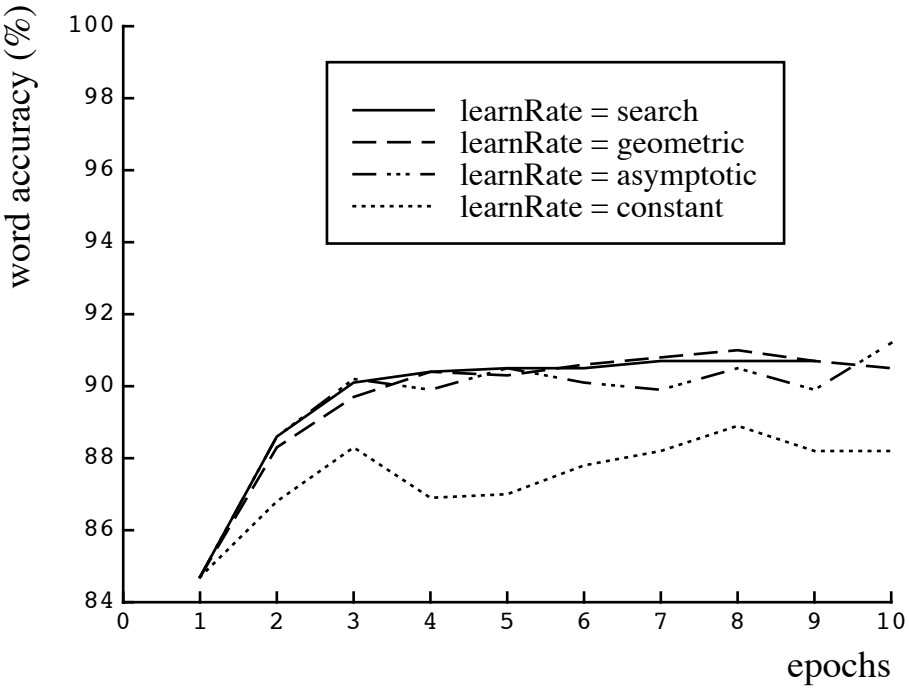


Figure 7.18: Performance of different types of learning rate schedules: Search is reliably optimal.

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.)

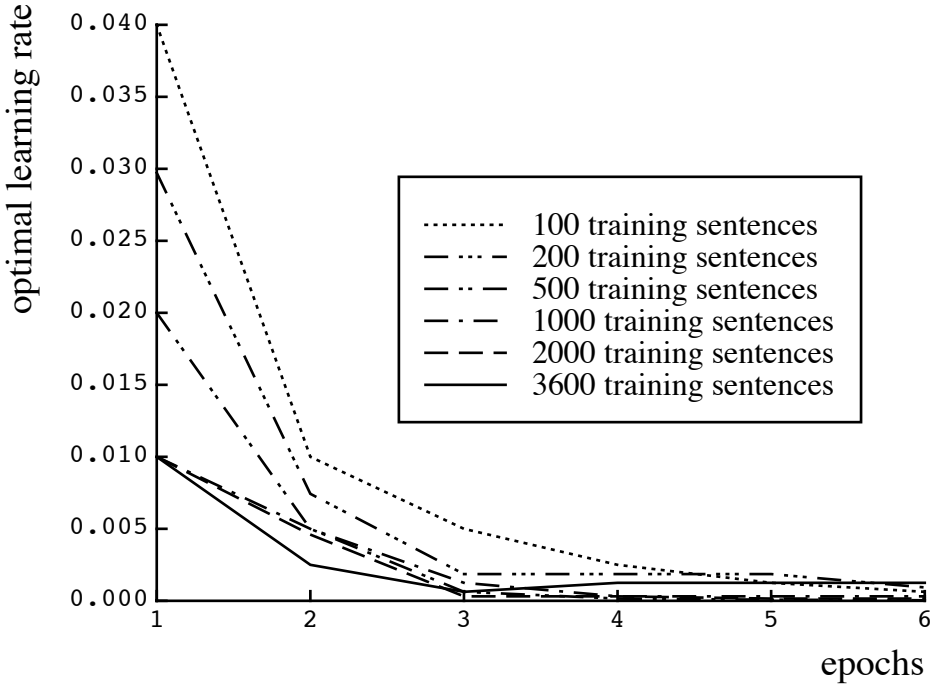


Figure 7.19: Learning rate schedules (as a function of training set size), as optimized by search.

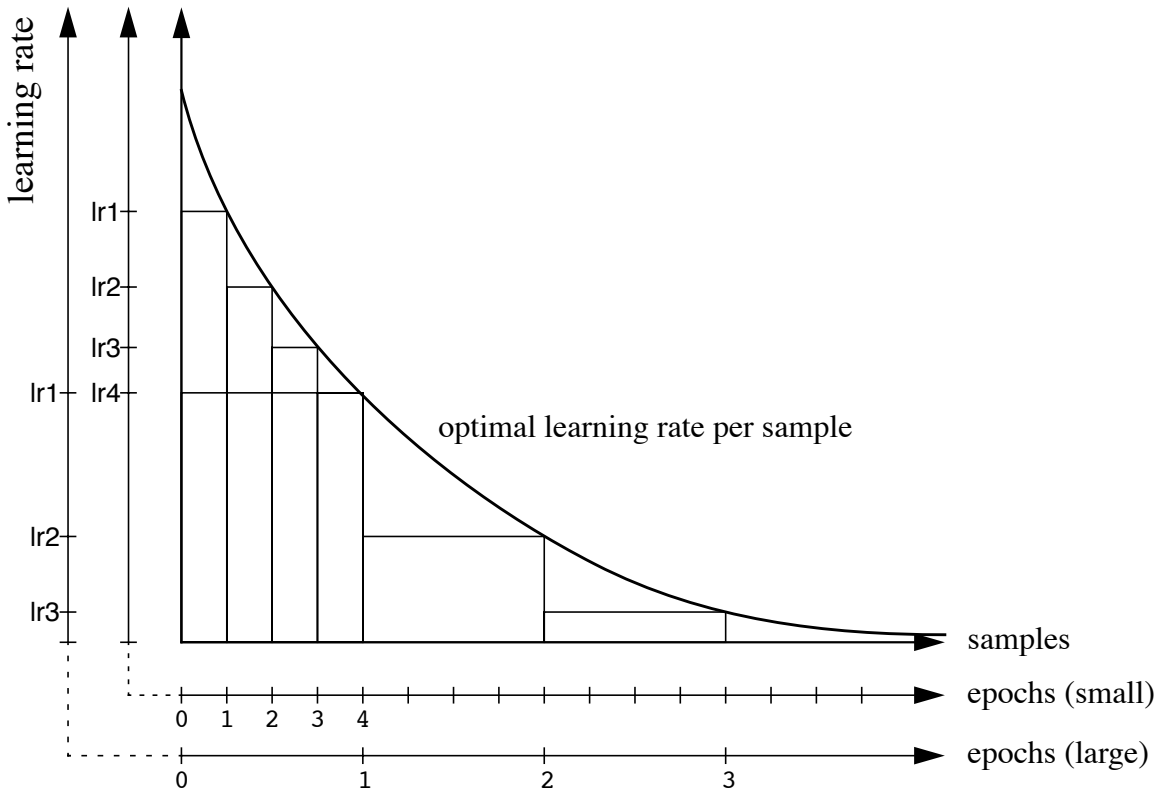


Figure 7.20: A larger training set should use smaller learning rates per epoch.

- **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.)

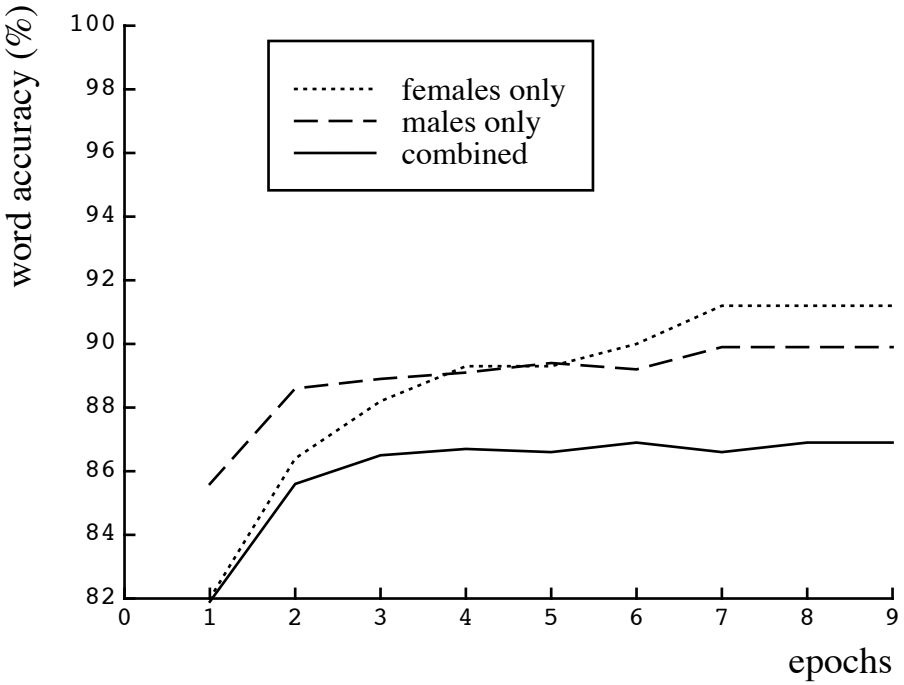


Figure 7.21: Gender dependent training improves results by separating two overlapping distributions.

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.

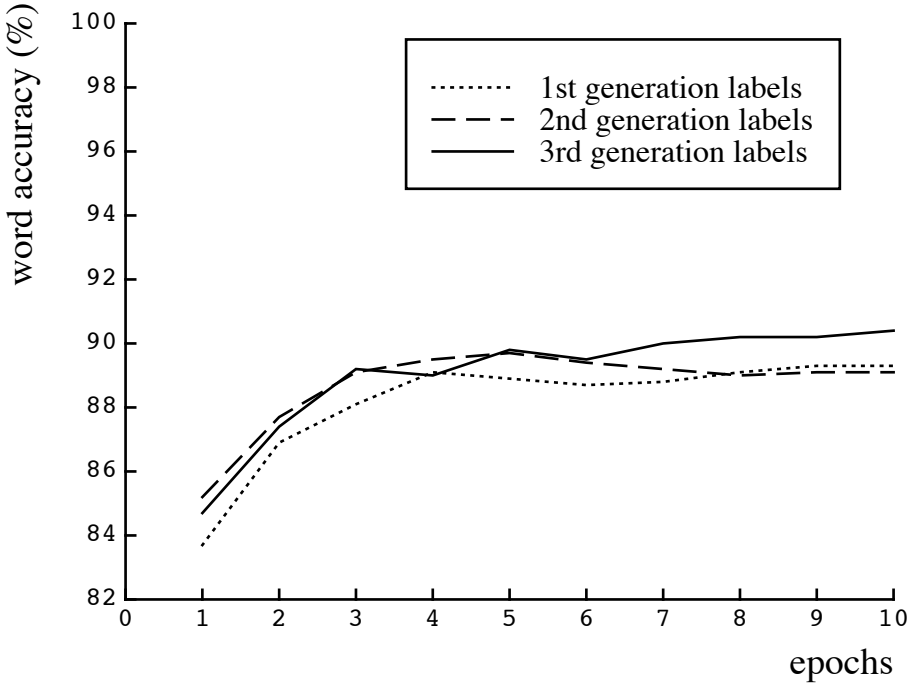


Figure 7.22: Recursive labeling optimizes the targets, and so improves accuracy.

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.

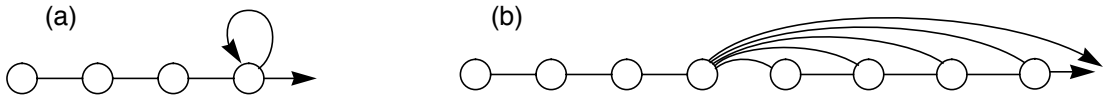


Figure 7.23: Duration constraints via state duplication. (a) 4 or more frames. (b) 4 to 8 frames.

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.

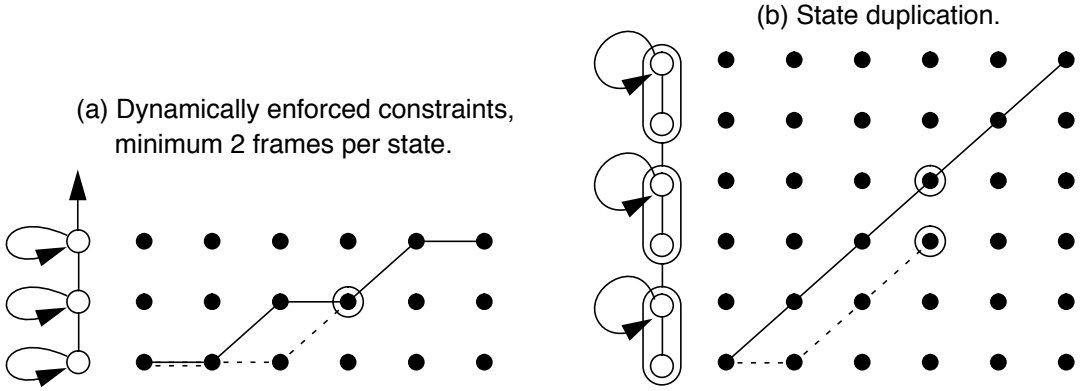


Figure 7.24: Minimum phoneme duration constraints. Only state duplication gives optimal results.

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).

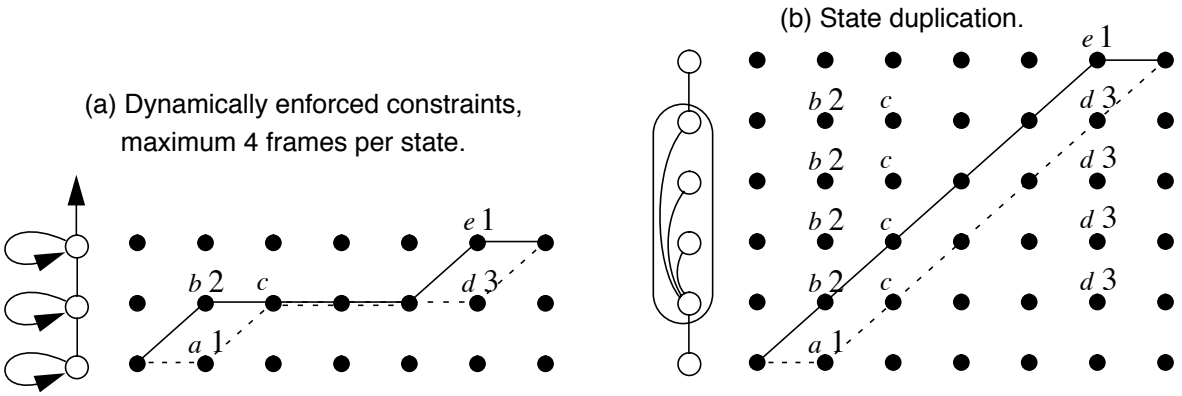


Figure 7.25: Maximum phoneme duration constraints. Only state duplication gives optimal results.

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 $\kappa \cdot \log P(dur)$,

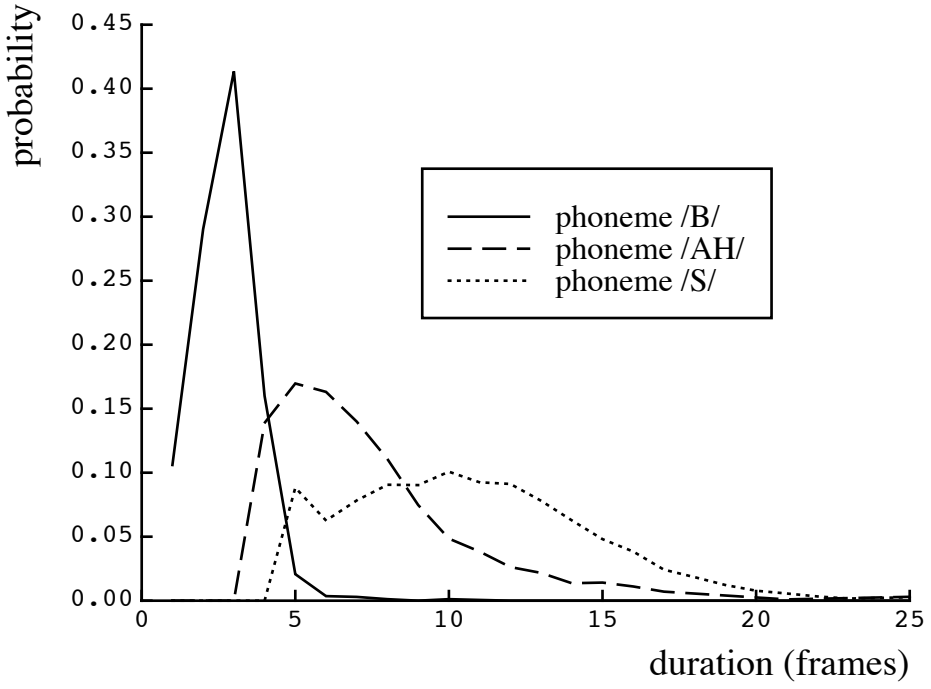


Figure 7.26: Histogram of durations of some typical phonemes.

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."

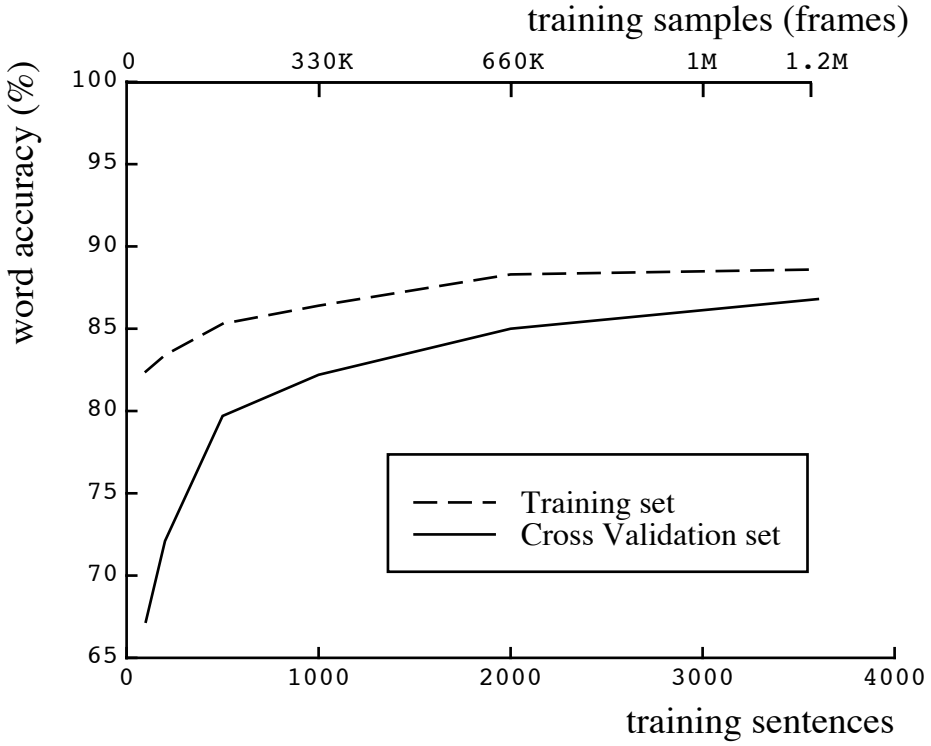


Figure 7.27: Generalization improves with more training data. The training-testing gap also shrinks.

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 con-

trast 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

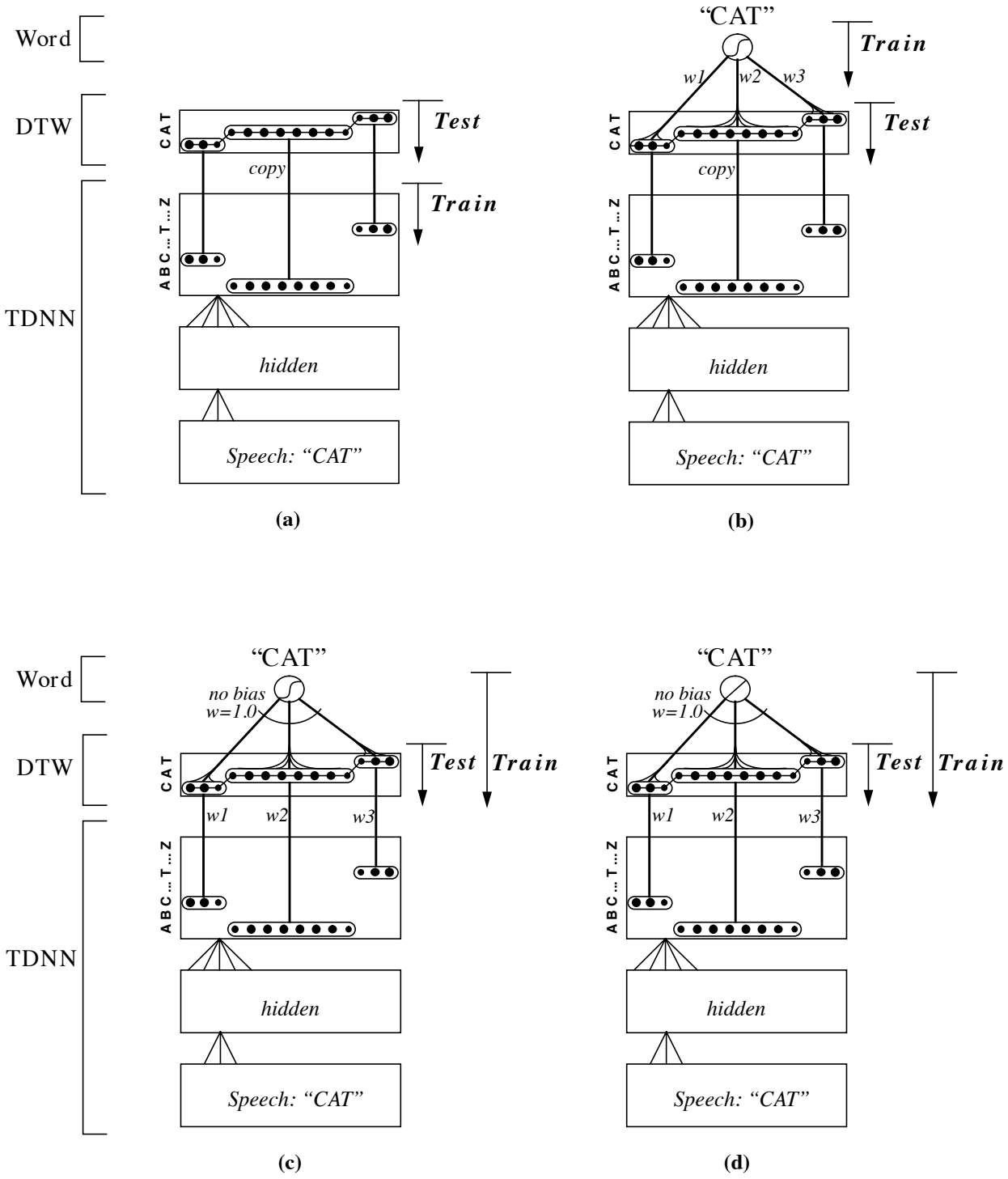


Figure 7.28: MS-TDNN, designed by resolving inconsistencies. (a) TDNN+DTW. (b) Adding word layer. (c) Pushing down weights. (d) Linear word units, for continuous speech recognition.

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.

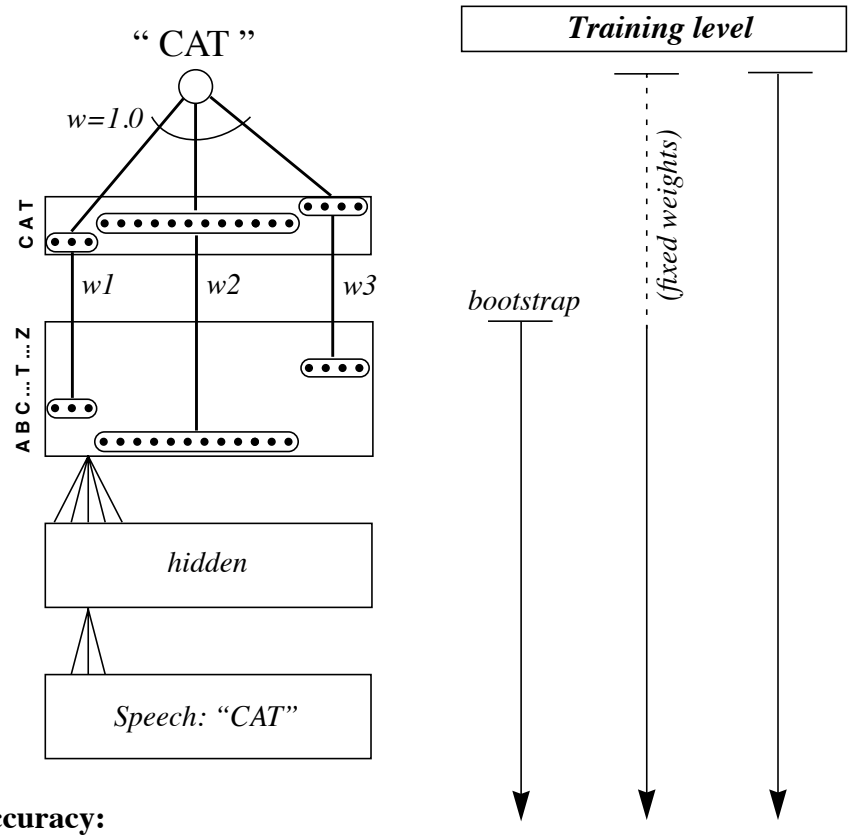
In our Resource Management experiments, we used an MS-TDNN with 16 LDA coefficients (with 9 time delays), 100 hidden units, 61 phoneme units, 6429 DTW units, and 994 word units, giving a total of 33,519 weights. The hidden layer used tanh activations, and the phoneme layer used softmax activations (preserving 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 → 400 hidden units	71.8%	90.9%	68%
DTW uses $Y_i \rightarrow \log(Y_i / (P(i)))$	74.9%	91.5%	66%
1 → 9 input frames	80.4%	89.6%	47%
state duplication	74.5%	86.0%	45%
normalized inputs	67.2%	77.9%	33%
batch → online weight updates	82.0%	88.0%	33%
asymmetric → symmetric sigmoids	72.9%	81.4%	31%
gender dependence	86.9%	90.6%	28%
constant → dynamic learning rates	88.2%	90.7%	21%
grouped → randomized sentences	79.0%	82.0%	14%
word level training	89.2%	90.5%	12%
1 → 3 states per phoneme	86.6%	88.2%	12%
recursive labeling	89.3%	90.4%	10%
FFT → 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.

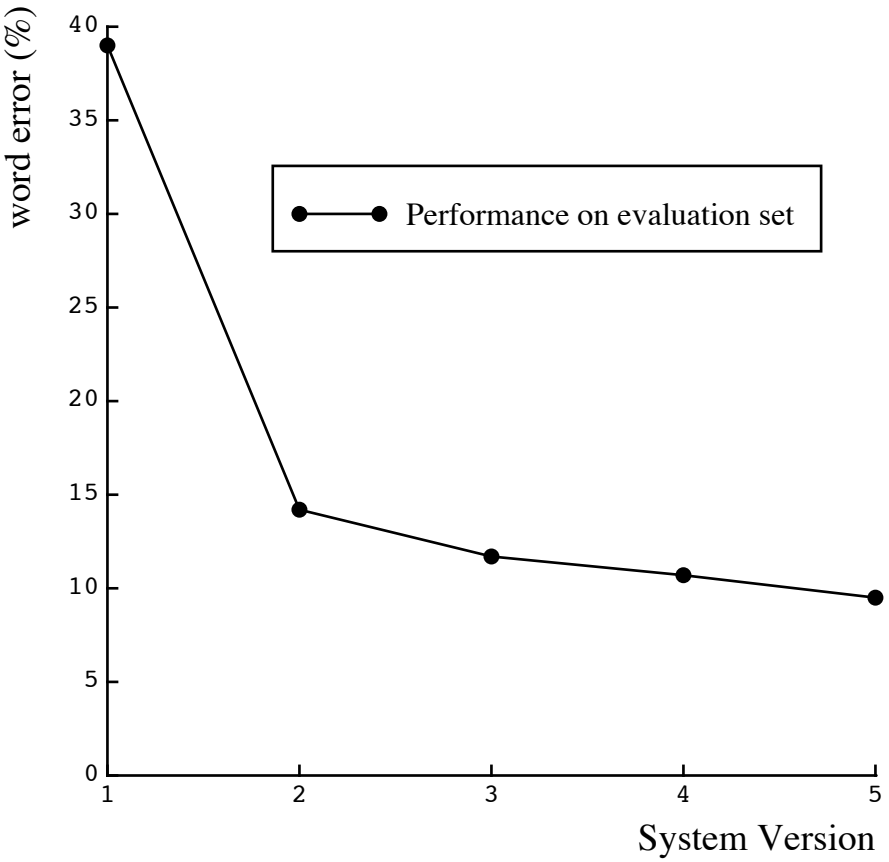


Figure 7.30: Snapshots of system performance on official evaluation set.

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 Network*s 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