Modular Neural Networks for Speech Recognition

Jürgen Fritsch

August 1996

CMU-CS-96-203

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Appeared as Diploma thesis in partial fulfilment of the degree Dipl-Inform. at the University of Karlsruhe, Germany.

Keywords: Hybrid Speech Recognition, Neural Networks, Hierarchical Mixtures of Experts, Mixture Models, Acoustic Modeling, Connectionist Context Modeling, Constructive Methods.

Abstract

In recent years, researchers have established the viability of so called hybrid NN/HMM large vocabulary, speaker independent continuous speech recognition systems, where neural networks (NN) are used for the estimation of acoustic emission probabilities for hidden Markov models (HMM) which provide statistical temporal modeling. Work in this direction is based on a proof, that neural networks can be trained to estimate posterior class probabilities. Advantages of the hybrid approach over traditional mixture of Gaussians based systems include discriminative training, fewer parameters, contextual inputs and faster sentence decoding.

However, hybrid systems usually have training times that are orders of magnitude higher than those observed in traditional systems. This is largely due to the costly, gradient-based error-backpropagation learning algorithm applied to very large neural networks, which often requires the use of specialized parallel hardware.

This thesis examines how a hybrid NN/HMM system can benefit from the use of modular and hierarchical neural networks such as the hierarchical mixtures of experts (HME) architecture. Based on a powerful statistical framework, it is shown that modularity and the principle of divide-and-conquer applied to neural network learning reduces training times significantly. We developed a hybrid speech recognition system based on modular neural networks and the state-of-the-art continuous density HMM speech recognizer JANUS. The system is evaluated on the English Spontaneous Scheduling Task (ESST), a 2400 word spontaneous speech database.

We developed an adaptive tree growing algorithm for the hierarchical mixtures of experts, which is shown to yield better usage of the parameters of the architecture than a pre-determined topology. We also explored alternative parameterizations of expert and gating networks based on Gaussian classifiers, which allow even faster training because of near-optimal initialization techniques. Finally, we enhanced our originally context independent hybrid speech recognizer to model polyphonic contexts, adopting decision tree clustered context classes from a Gaussian mixtures system.

Acknowledgments

I would like to thank Alex Waibel for his support and for the opportunity to work for almost one year in a progressive environment on a challenging project at Carnegie Mellon University, Pittsburgh. Also, I'd like to thank Michael Finke, who was in charge of supervising this thesis and who, in my opinion, did a pretty darn good job in doing so. I'm especially thankful for the inspiring discussions we were having every so often. I'm still trying hard to get him to accept an invitation for dinner. I also wish to thank all the other members of the nnspeech group for support and for having a good time. Last, but by no means least, I'd like to thank my parents for giving me all the support (including financial) and encouragment in my plans to experience the US and study at CMU.

Pittsburgh, July 1996

Contents

List of Figures

Chapter 1

Introduction

Speech is the natural form of communication for humans. We are using it excessively in our everyday life without noticing the complexity of this form of communication. Speech production is a highly nonlinear process that is strongly influenced by factors such as regional dialects, age, gender and emotional state. Speech perception is even more complex, since it involves a high degree of variability through additional background noise, different room acoustics and/or transmission characteristics in case of telephone lines. Despite this immense variability, we are able to use this form of communication even in adverse environments such as noisy parties. In fact, speech is the first and most natural way of communication, that we humans learn in the very beginning of our life.

In contrast, communicating with a computer requires knowledge about how to use a mouse and a keyboard and how to interpret textual messages appearing in lots of different windows. Most people would prefer to use speech when dealing with machines and computers. Some applications such as information systems over telephone lines even require this form of communication. There are lots of other applications where the users hands are busy doing other things and speech is the only reasonable input modality. Think about computers in cars and airplanes.

Therefore, there has been a large amount of research in automatic speech recognition, understanding and translation since the early 1950's. Although researchers have demonstrated impressive results with state-of-the-art hidden Markov model based systems, today's speech recognition technology is still far away from being competitive with human skills. Current speech recognition systems perform very well in very specific and limited domains. Applying such systems to new domains usually leads to unacceptably low performance.

Automatic speech recognition has to be considered far from being a solved problem and further improvement may require new insights and the exploration of new paradigms. The question is, what makes humans so good in perceiving, recognizing and understanding speech? Unfortunately we are also far away from understanding the cognitive processes necessary to answer this question. What we do know is, that information processing in the human brain differs completely from the way this is done in traditional computers. The human brain features billions of small processing elements (neurons) that are interconnected in complex ways and are operating in parallel.

Researchers attempt to simulate this kind of information processing in a very simplified way in form of *artificial neural networks*. Despite their simplicity, these networks have been applied succesfully to static pattern recognition, very often improving performance over traditional methods. They have also been used for the recognition of speech sounds, though it is still an open question how to apply them to temporal modeling necessary for continuous speech recognition. Since neural networks are very effective models for the discrimination of speech sounds, researchers started to build *hybrid systems* that combine the advantages of neural networks and hidden Markov models by replacing the usual parametric density modeling by discriminative artificial neural networks. Such systems have recently began to be competitive and sometimes superior to traditional speech recognition systems.

Mostly, neural networks are designed with parallel processing elements in mind, but implemented on standard serial computers. Also, they are considered to be one big monolithic entity that is trained and tested as a unit. This renders the learning process computationally very expensive and takes orders of magnitude longer than training traditional density estimators for speech recognition. Recently, *modular* and *hierarchically organized* neural networks have been studied extensively in the neural network and machine learning community (e.g. Meta-Pi networks [18], Hierarchical Mixtures of Experts [26],[27]). In these networks, the overall recognition task is divided among several small sub-networks, so called *experts*. The experts decisions are integrated in a hierarchical way, yielding the overall network output. Training times for such *mixtures of experts* systems are usually much smaller than those for traditional monolithic neural networks.

In this thesis, we investigate modular neural networks for hybrid continuous speech recognition systems, showing that modularity on the network level is a well fitting concept for efficient *and* highly accurate neural network based speech recognition.

The thesis is organized as follows: Chapter 2 gives a short overview of traditional neural networks and their statistical interpretation. Chapter 3 reviews basic concepts in statistical continuous speech recognition and the extension to NN/HMM hybrid speech recognition. Chapter 4 introduces the hierarchical mixture of experts architecture and learning algorithms for this modular neural network. Chapter 5 gives a novel constructive algorithm for automatically growing a hierarchical network that improves performance over static hierarchies. Chapter 6 discusses how to model context dependent phones in hybrid NN/HMM systems and chapter 7 considers alternative parameterizations for sub-networks in hierarchical mixtures of experts and discusses advantages. Finally, chapter 8 evaluates a hybrid NN/HMM system based on hierarchical modular neural networks and the JANUS HMM speech recognizer that was developed as part of this thesis. Chapter 9 presents conclusions and discusses enhancements in future work.

Chapter 2

Neural Networks

This chapter will briefly review common neural network architectures as far as they are important for the remainder of this thesis. It finishes with an important section on the relationships between neural networks and statistical models.

2.1 Introduction

Artificial neural networks are a wide class of flexible nonlinear regression and classification models. They consist of a (sometimes large) number of processing nodes, called *neurons*, which are simple linear or nonlinear computing elements. These elements are interconnected in a variety of ways and often organized in layers. Fig. 2.1 shows a basic processing node or neuron.

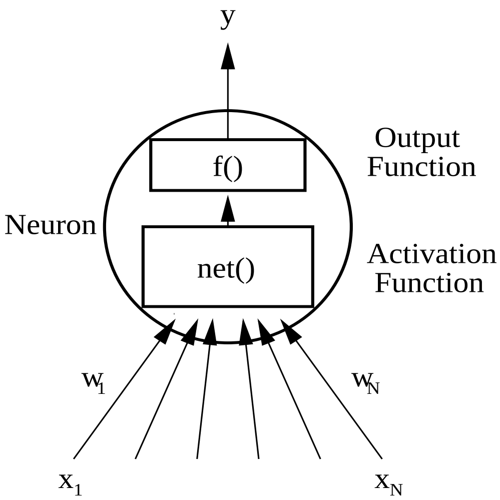

Figure 2.1: Processing element (neuron) in neural networks

It consists of an activation function $z = \mathrm{net}(x_1, \ldots, x_N) : \mathcal{R}^{\mathcal{N}} \to \mathcal{R}$ and a (possibly) non-linear output function $f(z) : \mathcal{R} \to \mathcal{R}$. The most common used activation functions are

$$\begin{aligned}\mathrm{net}(x_1, \ldots, x_N) &= \sum_{i=1}^{N} w_i x_i \\ \mathrm{net}(x_1, \ldots, x_N) &= \sum_{i=1}^{N} (x_i - w_i)^2\end{aligned}$$

Choices for the output function f are the identity, the *sigmoid* or the *softmax* function

$$f(z) = z \qquad f(z) = \frac{1}{1 + \exp(-z)} \qquad f(z_i) = \frac{\exp(z_i)}{\sum_{k=1}^{n} \exp(z_k)}$$

for a layer of n neurons. Associated with each neuron is a weight vector $\mathbf{w} = (w_1, \ldots, w_N)$. Sometimes, an additional *bias* weight w_0 with a fixed input value of 1 is used in order to extent the model from linear to affine transformations. Learning algorithms for neural networks estimate these weights (mostly) iteratively, in order to minimize a given error function of the outputs.

The most simple neural network architecture is a *perceptron* which may consist of just one neuron. It can be trained to discriminate between linearly separable classes using the sigmoid or softmax non-linearity as output function. However, for more complex discrimination or approximation tasks, networks with multiple layers of neurons are necessary. The next two sections describe the most commonly used neural network architectures for complex tasks and their learning algorithms.

2.2 Multi Layer Perceptrons

A *multi layer perceptron* (MLP) consists of several layers of neurons with full interconnections between neurons in adjacent layers (additional interconnections between non-adjacent layers are called shortcut connections). Fig. 2.2 depicts the structure of such an architecture. Input data is presented to the network at the input layer, which contains no processing nodes. It serves only as a data source for the following hidden layer(s). Finally, the networks output is computed by neurons in the output layer. The activation function of all neurons is the inner product between input and weight vectors. Only the activation of nodes in the input and output layers is directly observable. The nodes in hidden layers compute internal representations of the data.

MLP's are useful for supervised pattern recognition where the task is to learn a mapping between inputs $\mathbf{x}$ and outputs $\mathbf{t}$ given a set of training examples

$$\mathcal{T} = \{(\mathbf{x_1}, \mathbf{t_1}), \ldots, (\mathbf{x_N}, \mathbf{t_N})\}$$

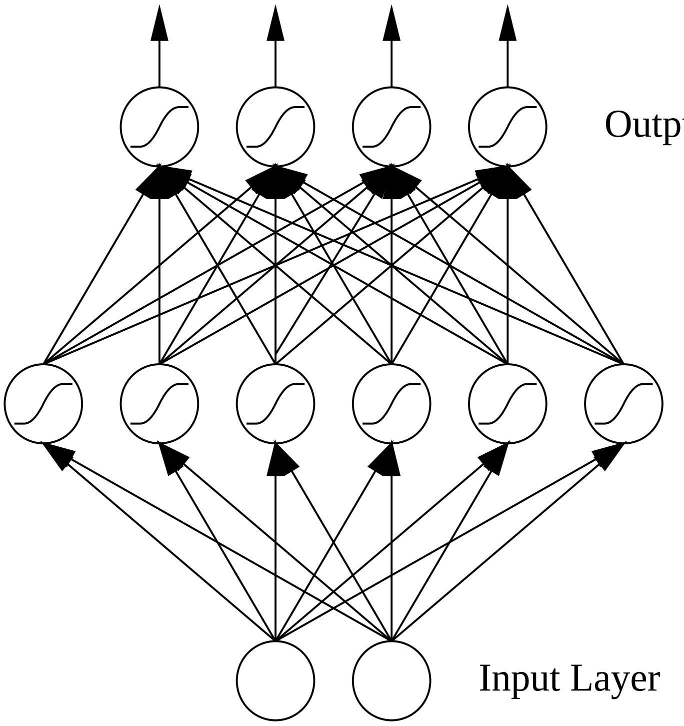

Figure 2.2: Multi Layer Perceptron (MLP)

In the training phase, the weights of an MLP are usually updated by an iterative learning algorithm called *error backpropagation*. After this procedure converges, the MLP can be used to map new (unseen) patterns.

The error-backpropagation learning algorithm is based on the chain rule for derivatives of continuous functions. The algorithm consists of a forward pass, in which training examples $\mathbf{x}$ are presented to the network and activations of output neurons $\mathbf{y}$ are computed. This is followed by a backpropagation step which updates the weights of neurons using the gradient of an error function such as the mean squared error or the cross entropy between network outputs $\mathbf{y}$ and given target outputs $\mathbf{t}$.

For example, using the mean squared error $E = 0.5 \sum_t \sum_i (y_i^{(t)} - t_i^{(t)})^2$ and the sigmoid output function $f(y_i) = 1/(1 + \exp(-z_i))$ with $z_i = \sum_j w_{ij} h_j$ where h_j are the activations of the hidden layer, the gradient with respect to the weights of neurons in the output layer $\mathbf{w_{ij}}$ is

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \sum_t (y_i^{(t)} - t_i^{(t)}) y_i^{(t)} (1 - y_i^{(t)}) h_j^{(t)} \\ &= \sum_t \delta_i^{(t)} h_j^{(t)} \end{aligned}$$

Thus, weights in the output layer can be updated as follows in order to minimize the error function:

$$w_{ij}^{(m+1)} = w_{ij}^{(m)} - \eta \sum_t \delta_i^{(t)} h_j^{(t)}$$

The derivative of the error function with respect to weights in the hidden layer $\mathbf{w_{jk}}$ can be computed using the chain rule which yields

$$\begin{aligned}\frac{\partial E}{\partial w_{jk}} &= \sum_t \sum_i (\delta_i^{(t)} w_{ij}) h_j^{(t)} (1 - h_j^{(t)}) x_k^{(t)} \\ &= \sum_t \delta_j^{(t)} x_k^{(t)}\end{aligned}$$

This leads to the following update rule for weights in the hidden layer:

$$w_{jk}^{(m+1)} = w_{jk}^{(m)} - \eta \sum_t \delta_j^{(t)} x_k^{(t)}$$

It is easy to generalize the backpropagation algorithm to networks with more than one hidden layer of neurons. It should be noted that there are lots of extensions of the basic algorithm such as an additional momentum term which aim at improving convergence speed and final performance. Nevertheless, the backpropagation algorithm is computationally very expensive, especially for large MLP's.

It can be shown that MLP's with at least one hidden layer can approximate any continuous function to any desired degree of accuracy, if there are enough hidden neurons available (this property is called *universal function approximation*). Thus, MLP's with one hidden layer are sufficient, although additional hidden layers may improve performance over single hidden layer networks with an equal number of neurons through increased model complexity.

2.3 Radial Basis Function networks

In Radial Basis Function networks (RBF), the hidden layer neurons compute radial basis functions of the inputs, similar to kernel functions in kernel regression. RBF networks consist of input, one hidden and output layer. The activation function of hidden neurons computes the *Euclidean* or *Mahalanobis* distance d between input and weight vectors. Usually, the output function of hidden layer neurons is

$$h_i = \exp(-\frac{d^2}{2})$$

The output layer neuron's activation function is the same as the one used for MLP's, the inner product of input and weight vector (with an additional bias input). The RBF

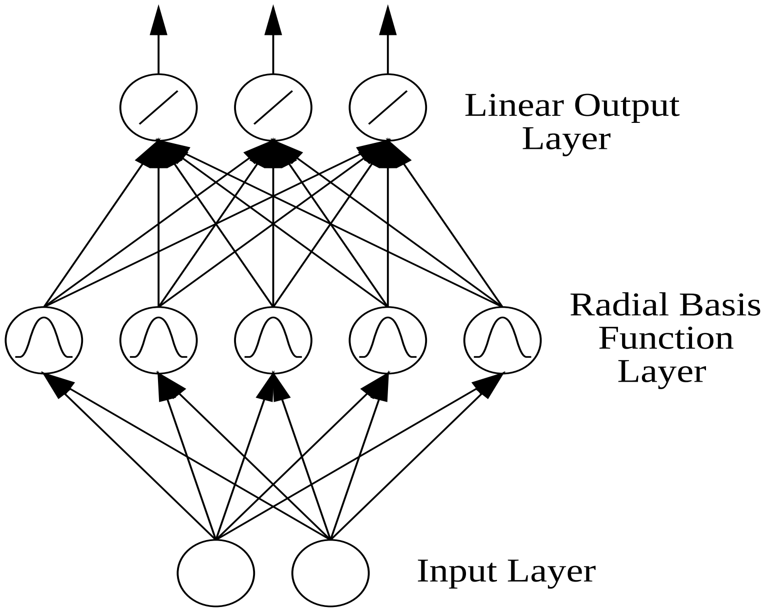

Figure 2.3: Radial Basis Function (RBF) network

network is mostly used for regression with a linear output layer although it is also possible to use it for classification with a sigmoid or softmax output layer.

Fig. 2.3 shows the structure of a RBF network. RBF hidden neurons are often called *localized receptive fields* because of the special form of their activation function. Sometimes the outputs of the hidden layer neurons are normalized to sum up to one as in kernel regression.

Training of RBF networks proceeds in two steps:

1. **RBF estimation for hidden neurons** Input feature vectors are clustered according to the desired number of hidden neurons using a procedure such as k-means, LBG or neural gas. This results in a set of RBF centers. If the model assumes a bandwidth, variance vector or covariance matrix for the hidden neurons, these parameters may be estimated using the data within each cluster.

2. **Linear Least Squares for output weight matrix** Once the parameters of the hidden neurons are computed, they remain fixed and the estimation of the weights of the (linear) output neurons reduces to a linear least squares problem which can be solved by the standard matrix inversion algorithm.

RBF networks can be trained much faster than MLP's, but it was shown that kernel methods such as RBF networks tend to require larger sample sizes to achieve the same performance, especially in high dimensional feature spaces.

2.4 Statistical Interpretation

Neural networks and statistical models are not competing methodologies for data analysis. There is considerable overlap between the two fields. Statistical methodology is directly applicable to most neural network models, resulting in more efficient parameter estimation and optimization (learning) algorithms. Additionally, statistical methods provide diagnostic tools such as confidence intervals and hypothesis testing which are missing in the field of neural networks.

Recently, statisticians published works which established ties between statistics and neural networks, sometimes showing the equivalence of statistical and neural network models. Sarle [48] shows relationships between many neural networks and statistical models and translates the jargons in the two fields. Ripley [47] provides a very interesting overview of the similarities of neural networks and statistical models.

2.4.1 Perceptrons

A perceptron with a linear output function computes a linear combination of the input features. It is nothing else but a linear regression model that can be fit most efficiently by linear least squares.

In case the output function is nonlinear, a perceptron is a *generalized linear model* (GLIM) with the exception that for a perceptron, the nonlinearity is mostly chosen ad hoc, while the nonlinearity of a GLIM is fixed, once a probabilistic model of the outputs given the inputs is chosen. GLIM's are fitted by maximum likelihood methods for a variety of distributions of the exponential family. For multiway classification, one usually assumes a multinomial (Poisson) density model, which forces the use of the softmax nonlinearity as output function for the GLIM/perceptron. It is considerably more effective to use maximum likelihood fitting than mean square error minimization to estimate the parameters of a perceptron. This fact is important for modular neural networks with simple perceptron-like processing elements, such as the architecture that we will introduce later in this thesis.

2.4.2 Multi Layer Perceptrons

Like a perceptron, a MLP has counterparts in statistics as well, depending on the number of hidden layers and the number of neurons in the hidden layers. Sarle [48] categorizes MLP's into the following three groups:

- **Small number of hidden neurons.** MLP can be considered as a parametric model such as polynomial regression.

- **Moderate number of hidden neurons.** MLP can be considered a quasi-parametric model similar to projection pursuit regression.

- **Large number of hidden neurons,** possibly increasing with the sample size. MLP can be considered as a nonparametric sieve.

It is this smooth transition between parametric and nonparametric models that renders MLP's especially useful. The error-backpropagation learning algorithm for MLP's is iterative, slow and requires the careful adaptation of various learning parameters such as the learning rate and the momentum factor by trial and error. Since MLP's perform multivariate multiple nonlinear regression, its parameters may be estimated much more efficiently using nonlinear optimization algorithms such as those used for projection pursuit models.

2.4.3 Unsupervised Learning

Unsupervised learning for neural networks consists in extracting useful features from the input data and eliminating redundancy, without having any target or output vectors associated with each input vector. From a statistical point of view, things are different. The goal in most forms of unsupervised learning is to estimate feature variables from which the observed data can be predicted. In this formulation, the observed data is considered to be both input and target of the learning process.

Unsupervised Hebbian learning for a one layer linear network, for example, is identical to principal component analysis, which provides the optimal transformation matrix. This fact is well-known from statistical theory and many variations of Hebbian learning consist of inefficient approximations of principal component analysis.

Chapter 3

Hybrid Speech Recognition

This chapter will first review the basic concepts of today's state-of-the-art speech recognition technology based on *hidden Markov models*. It will then discuss advantages and drawbacks and shortcomings of this approach which motivate *hybrid speech recognition* systems. The term *hybrid speech recognition systems* is now widely used for systems that try to bring together the best of two worlds: Statistical time alignment by hidden Markov models and discriminative observation probability estimation by neural networks instead of by means of parametric multimodal distributions. We will briefly discuss two such systems, one based on the multi layer perceptron (MLP) and one based on recurrent neural networks (RNN), as they are currently being investigated by researchers in the speech community. Finally, we will discuss problems observed with large monolithic neural networks as used in practical implementations of hybrid speech recognition systems, motivating the exploration of modular and hierarchical neural networks for hybrid speech recognition.

3.1 Speech Recognition

This section gives a quick overview of current hidden Markov model (HMM) based speech recognition technology as it is used in almost all current state-of-the-art speech recognition systems. Readers already familiar with these concepts may want to skip to the next section.

3.1.1 Overview

Fig. 3.1 shows the basic setup of a speech recognition system revealing all its major components.

Input to the system is a sampled waveform of the audio signal as recorded by a microphone. Note that the room characteristics, the kind of microphone and A/D transducers

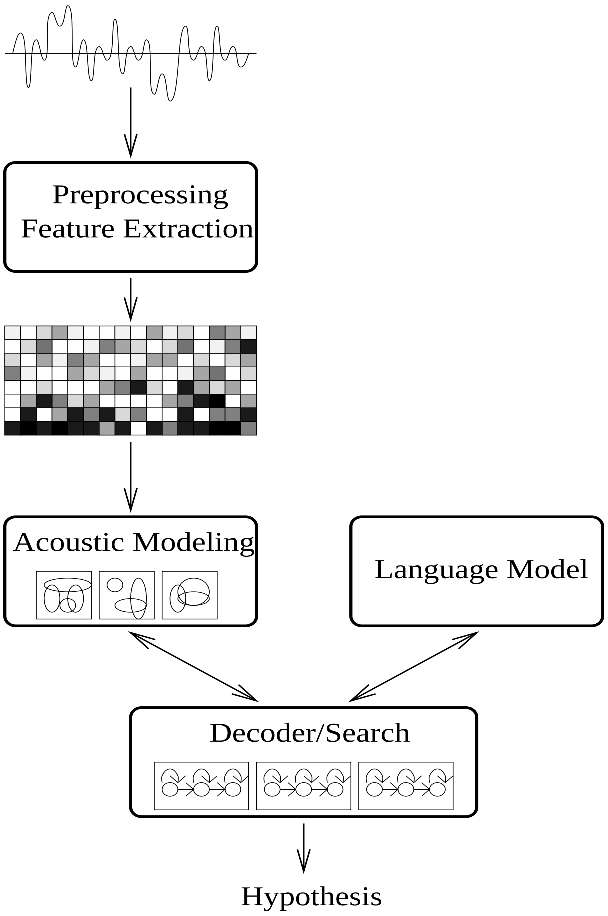

Figure 3.1: Overview: Automatic Speech Recognition

that are used to record the audio signal can have a severe effect on the speech representation and recognition. Recently, large efforts have been put into developing so called *robust* systems, which tolerate different kinds of microphones, room characteristics and noise conditions in the *prepocessing* stage. This stage is sometimes called *feature extraction* or *front-end*. It computes a sequence of features, mostly derived from spectral or cepstral representations of speech, which are more suitable for the following stages than the raw speech waveform. The *acoustic modeling* stage models a set of speech sounds by hidden Markov models and (mostly) continuous parametric distributions. For any given observation at any time step, the acoustic modeling stage provides local probabilities for each of the modeled atomic sound units. These local scores are then used in a dynamic programming search (*decoder*) stage, to determine the most likely sequence of words, given the acoustics. Additional information about prior probabilities of sequences of words is supplied to the decoder by the *language model*. We will now go into some details, concerning the basic blocks of a speech recognizer, but we can not provide an exhaustive overview of this field. See [52],[32],[51] for additional information.

3.1.2 Preprocessing

Speech signals have been observed to have stationary properties over periods no longer than about 20ms. Therefore, most speech recognition frond-ends use a sliding window of between 5ms-20ms to extract a vector of features from the speech waveform. Such vectors are called *frames* and are typically extracted at a rate of about 100Hz. The ultimate preprocessing stage should generate a representation of the speech signal, that (1) compresses the speech signal as far as possible, without loosing any information necessary for the recognition afterwards and (2) facilitates discrimination between different speech sounds. Fig. 3.2 shows the sequence of operations usually applied to the speech waveform in order to compute spectral or cepstral features.

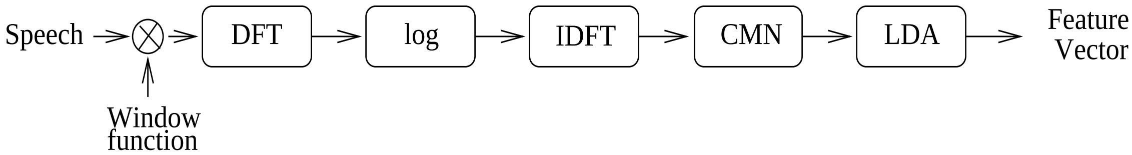

Figure 3.2: Preprocessing for Speech Recognition

The speech signal is multiplied with a window function, then a discrete Fourier transform (DFT) and the power spectrum is computed. The cepstrum is computed by applying the logarithm and an inverse discrete Fourier Transform (IDFT) to the spectrum. Often, additional steps such as the following are applied:

- **CMN (cepstral mean normalization)** The idea behind this technique is, that the observed audio signal is a linear superimposition of speech and noise, which is preserved in the cepstral domain. By subtracting the cepstral mean over a whole utterance, the additive stationary parts of the cepstrum are removed.

- **LDA (linear discriminant analysis)** This technique has proven very useful to reduce the dimensionality of feature vectors. It applies a linear transformation that minimizes intra-class distance while maximizing inter-class distance. Dimensionality reduction is achieved by dropping coefficients in the resulting feature vectors according to their significance. Often, multiple frames are concatenated prior to the application of LDA to include contextual information to the resulting features.

- **VTLN (vocal tract length normalization)** Different speakers have different vocal tract lengths. Different vocal tract lengths imply different pitch and formant frequencies for different speakers. This is usually compensated by a linear or piecewise-linear warping of the frequency axis in the spectrum based on statistics of formant frequencies.

- **PLP (perceptual linear prediction)** Performs several psychophysically based spectral transformations. It is based on the all-pole filter model used in Linear Predictive Analysis (LPA).

3.1.3 Hidden Markov Models

HMMs model a sequence of observations (in our case a sequence of feature vectors) as a piecewise stationary process. A discrete HMM is a stochastic finite state automaton $\mathcal{A} = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \pi\}$ with a set of stationary *states* $\mathbf{S}$, a *transition probability* matrix $\mathbf{A}$, a *emission probability* matrix $\mathbf{B}$ and a set of *initial state probabilities* π. Usually, speech recognition systems use strictly left to right HMMs to model words, sylables, phonemes or sub-phonetic units. Often, words are modeled as a sequence of phonemes, which in turn are modeled as a sequence of HMM states. Fig. 3.3 shows the topology of a typical phoneme HMM.

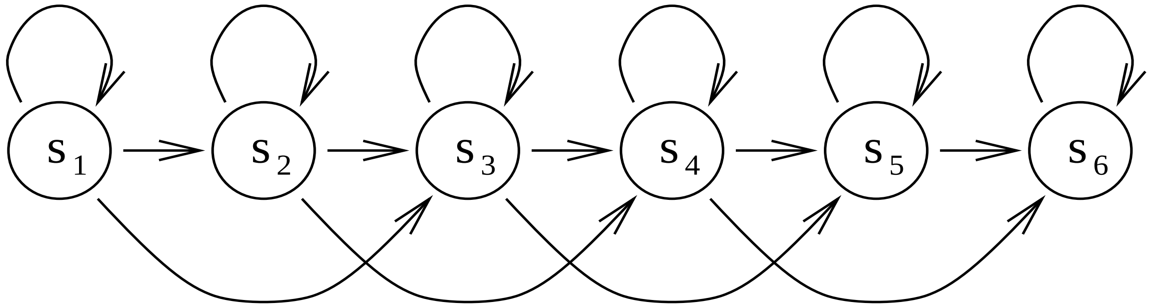

Figure 3.3: Hidden Markov Model topology for phonemes

Different states in a phonetic HMM model different stationary acoustic sounds at the beginning, middle and end of a phoneme. Viewing the HMM as a generative model, the term 'hidden' becomes clearer. HMMs consist of two concurrent stoachastic processes. One is the un-observable sequence of states that models the temporal structure of speech, the other is the observable sequence of emitted output symbols in each state, modeling the the locally stationary character of speech sounds. There are three problems arising, when using HMMs to model sequences of observations:

Evaluation What is the probability that a given HMM generated a given sequence of observations.

Decoding Given a sequence of observations and a HMM, what is the most likely sequence of states through the HMM that lead to the generation of the observations.

Parameter estimation Given a HMM and a set of observation sequences to be modeled by this HMM, how can we adapt the parameters (emission and transition probability distributions) of the model to maximize the likelihood of generation.

All of the above three problems have very efficient solutions in form of special cases of dynamic programming algorithms. For instance, the evaluation problem occurs in isolated word recognition where we want to score different word HMMs according to their likelihood. It can be solved using the *Forward* algorithm. The decoding problem occurs in continuous speech recognition where we are seeking the most probable path through a very large HMM consisting of all possible sequences of basic sound units. Once we found this path, we can derive the most probable sequence of phonemes or words. The decoding problem can be solved using the *Viterbi* algorithm. Fig. 3.4 shows a typical trellis diagram with the optimal path as a Viterbi algorithm would produce it. The diagram also shows all possible state transitions at one specific time point.

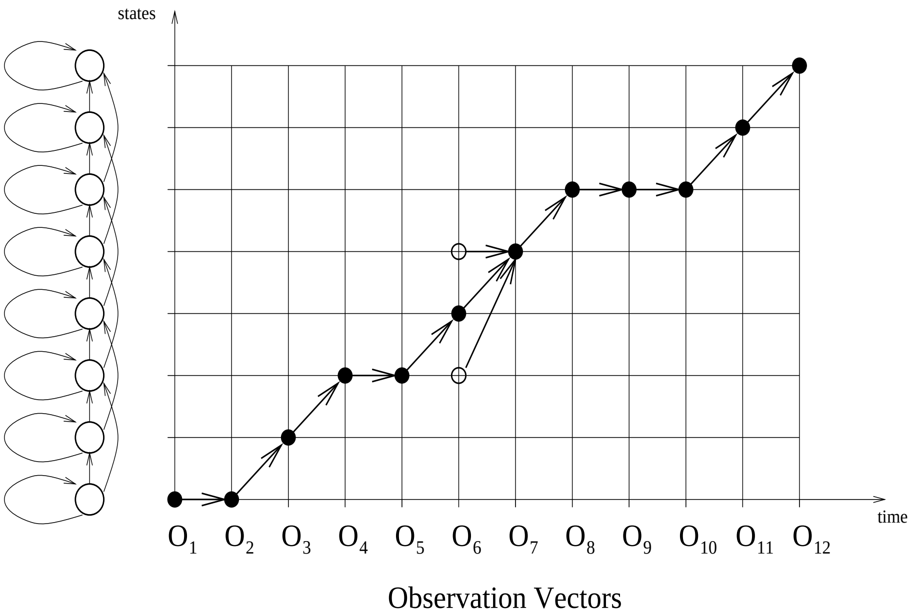

Figure 3.4: State trellis and the Viterbi algorithm

The last problem, also called the training problem, can be solved by the *Forward-Backward* or *Baum-Welch* algorithm, which is essentially a version of the *Expectation-Maximization (EM)* [10] algorithm. In the case of left-right HMMs with a constant small number of transitions in each state, all three algorithms have a computational complexity of only $O(NT)$, where N is the number of states in the HMM and T is the number of observations.

3.1.4 Acoustic Modeling

Today's state-of-the-art speech recognition systems use parametric multimodal probability densities to model continuous observations instead of discrete observations as required in the standard HMM. It was shown empirically, that such systems yield better performance than systems based on vector-quantization derived discrete observation symbols. Continuous densities are mostly modeled by mixtures of Gaussians, since it was shown that these mixture models can approximate any kind of distribution, given enough data to estimate its parameters reliably. In a continuous density HMM, the probability of observation vector $\mathbf{x}$ in a state s_i is modeled by

$$p(\mathbf{x}|s_i) = \sum_{j=1}^{N} c_{ij} N_{ij}(\mathbf{x}) \qquad \text{with} \qquad N_{ij}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_{ij}|}} \exp\{-\frac{1}{2}(\mathbf{x}-\mu_{ij})^T \Sigma_{ij}^{-1} (\mathbf{x}-\mu_{ij})\}$$

The Forward-Backward algorithm can be extended to continuous density observations which yields update formulas for the parameters c_{ij} (mixture weights), μ_{ij} (means) and Σ_{ij} (covariance matrices).

If there is not enough training data to estimate a separate mixture of Gaussians for each state of large HMMs, one can share parameters among different states, so that they use the same set of Gaussians but with different mixture weights. This form of parameter tying is known as *semi-continuous density modeling (SCHMM)*. For example, there is a special case of this kind of modeling, called *phonetically tied semi-continuous density modeling (PTSCHMM)* where all the states of a phonetic HMM share the same set of Gaussian densities. Other forms of parameter sharing include state clustering and/or decision tree clustering.

Another issue is the modeling of context-dependency on the HMM level. It was shown (see for example [32]) that the explicit modeling of phonemes in different contexts by different HMMs yields a vast improvement over context-independent systems. Current systems model biphone, triphone or even polyphone contexts to account for the variability of speech sounds in different contexts. Since the average number of monophones used in a typical system ranges around 50, n-phone contexts would require the modeling of 50^n different acoustic models. This clearly is not feasible in practice, especially since many contexts occur rarely or even never in a given training corpus. The solution to this problem is the use of decision tree's with a set of phonetic context questions to cluster the polyphonic contexts into a reasonably small set of context classes, which are then modelled by separate HMM's. See [44] for an introduction to decision trees.

3.1.5 Decoding/Search

The decoder is the essential recognition part of a speech recognizer. It uses locally computed emission probabilities to find the most likely sequence of words in a dynamic programming fashion. Typical large-vocabulary continuous-speech recognition tasks today involve a vocabulary of 20k to 50k words. Additionally, context-dependent modeling yields over 10k of context-dependent phoneme models. Clearly, the standard Viterbi algorithm for finding the most likely sequence of HMM states is not applicable without modifications, because of the combinatorical explosion of the size of the search space. Therefore, most decoders are organized in a multi-pass strategy, applying more detailed models in succesive passes with restricted search spaces. Most decoders are based on either time-synchronous Viterbi beam search or stack decoding, which is essentially an A^* search.

Viterbi beam search is a modified Viterbi algorithm, where active states are pruned at each time step, based on either their cummulative score or on their ranking in a list sorted by cummulative score. This way, only a very limited number of state, phone and word transitions (50-200) are considered at each time step. A disadvantage of the Viterbi beam search is the time-synchronous left-to-right mode of operation which may lead to recognition errors because a lot of hypotheses are being pruned away based on just the beginning part of the actual utterance although the remaining part may suggest to keep the hypotheses.

A stack decoder is a non time-synchronous search algorithm, comparing incomplete paths of different lengths by means of a likelihood function that estimates the probability of the most likely remaining paths. The basic data structure used in this kind of search is a stack which contains a sorted list of active incomplete paths together with their score. At each iteration of the search, the top entry is examined and all possible extensions of the associated incomplete path are evaluated and inserted in the stack. The accuracy of this algorithm clearly depends on the size of the stack. Often, a stack decoder is used as a second search pass, following a Viterbi beam search that restricts the search space and provides estimates of probabilities of partial paths.

Other important search techniques, especially in the case of large vocabularies, include the organization of the pronunciation lexicon in form of a phonetic prefix tree. Since many words start with the same sequence of phonemes, the storage requirements can be reduced significantly using this approach.

Usually the output of the decoder is not only a single best scored hypothesis for a given utterance, but a list of the first n-best hypotheses or a word graph (word lattice) which can be subject to further processing.

3.1.6 Language Modeling

The task of automatic speech recognition is to find the most probable word sequence $\mathbf{w}$ given a sequence of acoustic observations $\mathbf{x}$, which is the maximum posterior sentence probability. According to Bayes rule, it can be decomposed into

$$\max_i p(\mathbf{w}_i|\mathbf{x}) = \max_i \frac{p(\mathbf{x}|\mathbf{w}_i)P(\mathbf{w}_i)}{p(\mathbf{x})}$$

The denominator can be neglected since it is constant for all $\mathbf{w}_i$ and $p(\mathbf{x}|\mathbf{w}_i)$ is computed by the acoustic model. It remains to provide a means for estimating prior sentence probabilities $P(\mathbf{w}_i)$. These probabilities are computed by the language model and can be used in the decoder and/or in subsequent rescoring passes based on n-best lists or word graphs.

We will not go into much detail here and only describe the very basics of language modeling, that is, statistical n-gram modeling. The basic assumption here, is that probabilities of words in a sentence are only depending on the previous $n-1$ words. The prior probability of a given sentence can then be factored as follows:

$$P(\mathbf{w}) = \prod_{k=1}^{m} P(w_k|w_{k-1},\ldots,w_1) \approx \prod_{k=1}^{m} P(w_k|w_{k-1},\ldots,w_{k-n+1})$$

In case of a bigram model, we have to estimate probabilities $p(w_k|w_{k-1})$, in case of a trigram model, we have to estimate probabilities $p(w_k|w_{k-1},w_{k-2})$. This can be done by scanning large text corpora and counting occurances of word pairs or word triples, respectively. Since many trigrams that may be encountered in a test sentence do not occur in even the largest text corpora, we have to use a smoothing technique which avoids word probabilities of zero. The standard procedure here is to use a weighted sum of unigram, bigram and trigram probabilities where the weights are determined by an algorithm called deleted interpolation. Despite the simplicity of this approach, it was proven to work very well for large vocabulary continuous speech recognition.

3.2 Discussion

This section discusses advantages and drawbacks of the traditional HMM based speech recognition systems, as they have been described in the previous sections.

Advantages:

- **Rich mathematical framework** HMM's are based on a flexible statistical theory which allows to build even large systems consistently.
- **Efficient learning and decoding algorithms** These algorithms handle sequences of observations probabilistically and they do not require an explicit hand segmentation in terms of the basic speech units. They can be implemented very efficiently even for very large systems.
- **Easy integration of multiple knowledge sources** Different levels of constraints (e.g. phonological and syntacical) can be incorporated within the HMM framework as long as these are expressed in the same in terms of the same statistical formalism.

Disadvantages:

- **Poor discrimination** Estimation of the parameters of HMM's is based on likelihood maximization. This means, only correct models receive training information, incorrect models do not get any feedback.

- **1st order Markov assumption** Current observations and state transitions are depending only on the previous state – all other history is neglected.

- **Independence assumptions** Consecutive feature vectors are assumed to be statistically independent.

- **Require distributional assumptions** For example, modeling acoustic observations by mixture of Gaussians with diagonal covariance matrices requires uncorrelated feature coefficients, which is not the case.

- **Assumption that speech is a piecewise stationary process** All representational power goes into the modeling of stationary parts of speech, although it is known that speech should rather be modeled as a sequence of transitions or trajectories in the feature space. This is somehow alleviated by incorporating delta and delta delta features into the process of feature generation.

- **Assumption of exponential state duration distributions** This assumption is an integral part of 1st order HMM's. It can only be circumvent by applying explicit state duration modeling, that is, imposing external duration distributions such as a gamma distribution.

- **Maximum likelihood based** This is a disadvantage because maximum likelihood estimation always relies on the correctness of the models which is simply not true in the case of speech recognition.

- **Complexity** All of the above disadvantages require additonal modifications and enhancements of the basic HMM technology that lead to complex heuristics based systems.

3.3 Hybrid Speech Recognition

Hybrid speech recognition systems try to attack some of the disadvantages of traditional HMM's while still adhering to the general statistical formalism. In particular, since these methods use neural networks as emission probability esimators, training is based on posterior class probabilities instead of maximum likelihood. Neural network classifiers are discriminative in nature and do not impose constraints such as uncorrelated feature coefficients although they are not free of distributional assumptions as shown in the previous chapter.

3.3.1 Neural Networks as Statistical Estimators

It was shown that neural networks such as MLP's can be trained to compute estimates of the posterior class probabilities $p(\omega_i|\mathbf{x})$, given an input vector. HMM's require the computation of likelihoods $p(\mathbf{x}, q_i)$ for hypothesized states q_i. Fortunately, we can apply Bayes rule to convert posteriors into scaled likelihoods that can then be used as observation probabilities:

$$p(\mathbf{x}|q_i) = c\frac{p(q_i|\mathbf{x})}{P(q_i)}$$

In the above equation, $P(q_i)$ is the prior probability of state q_i and the neural network must be trained to produce estimates of posterior state probabilities $p(q_i|\mathbf{x})$. This means, we need to train a neural network which has as many output nodes as there are HMM states. We can compute scaled likelihoods by dividing the network outputs by the prior state probabilities.

It should be noted that in theory, HMM's could also be trained using local posterior probabilities as emission probabilities. In [2], an iterative procedure based on the EM algorithm is used to compute local estimates of posterior class probabilities which can be used as 'soft' targets for neural networks. This approach aims at optimizing the global posterior probability for the sequence of word models, instead of maximizing the likelihood.

To keep the number of states low enough to train a large neural networks in a reasonable amount of time, most researchers first experimented with context-independent HMM systems with one-state phonemic HMM's. In this case, the number of HMM states equals the number of phonemes and the neural network estimates posterior phoneme probabilities. The extension of this technique to context-dependent modeling is possible by factoring context-dependent posteriors and using multiple neural networks to estimate context-dependent observation probabilities. This will be described in detail in a separate chapter (6).

3.3.2 Training Issues

In order to train a neural network such that the resulting outputs estimate posterior class probabilities, we need to generate target vectors for each frame. When training the network on 1-out-of-N targets, an explicit segmentation in form of class-labels for each frame is necessary. Usually these labels are generated by an existing HMM speech recognizer for the given task. For any given training utterance, there is a sentence transcription available. This transcription is used to build a sentence HMM model by concatenating the HMM's of the corresponding word models, which in turn, are build by concatenating subword-unit HMM models with respect to the word pronunciation dictionary. Once a HMM model for the complete utterance is built, we can do a forced Viterbi alignment using the existing recognizer, which gives us the most probable sequence of states through the HMM, given the sequence of acoustic observations. Thus, we have generated state labels for each frame of the utterance. Once a neural network is sufficiently trained on these targets, using the performance on an independent cross validation set as a measure of generalization, new targets can be computed by recomputing the forced Viterbi alignment using the neural network to compute emission probabilities. This procedure may continue in an iterative manner. Alternatively, the Forward-Backward instead of the Viterbi alignment algorithm may be used which will result in soft targets.

3.4 Examples of Hybrid Systems

This section will briefly describe two current hybrid systems that have been succesfully used for continuous speech recognition. One is based on large multi layer perceptrons (MLP), the other uses recurrent neural networks (RNN).

3.4.1 A MLP based Hybrid

Researchers at the International Computer Science Institute (ICSI) in Berkeley have developed a hybrid speech recognition system that uses large multi layer perceptrons (MLP) to estimate posterior class probabilities. Fig. 3.5 shows an example of such a network.

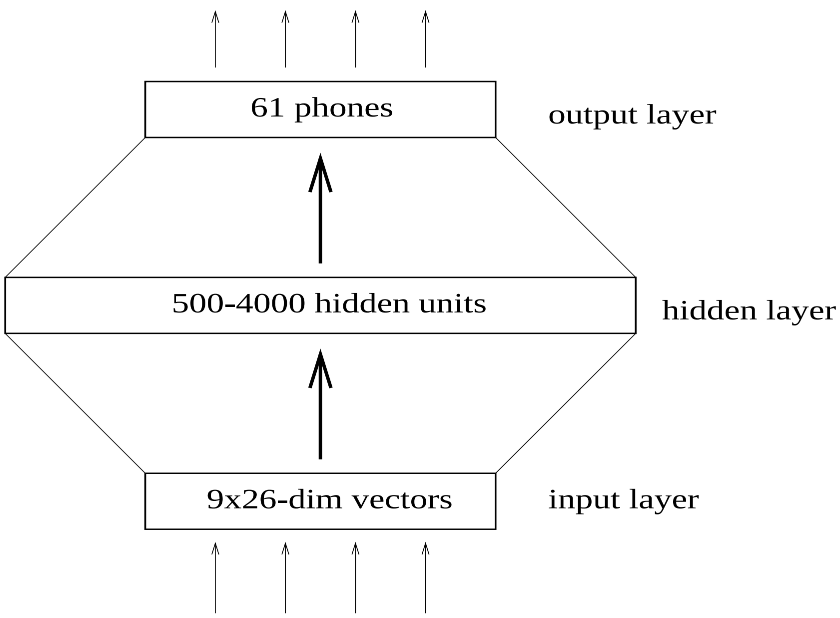

Figure 3.5: ICSI's multi layer perceptron topology

The network is trained by stochastic gradient error backpropagation using the Ring Array Processor (RAP), a parallel computer needed to keep training times in a reasonable range (days and not weeks). To reduce training times even further, the network was initialized by training on a hand-labeled phonetic database (TIMIT) before training it on the larger target task.

3.4.2 A RNN based Hybrid

The group at Cambridge University Engineering Department (CUED) has developed a hybrid connectionist/HMM speech recognition system called ABBOT [21], which uses recurrent neural networks to compute emission probabilities. The network is depicted in Fig. 3.6. It uses a set of state units that have recurrent connections from their outputs back to their inputs (these units also have connections to the input nodes). State units and input nodes are connected to the output layer.

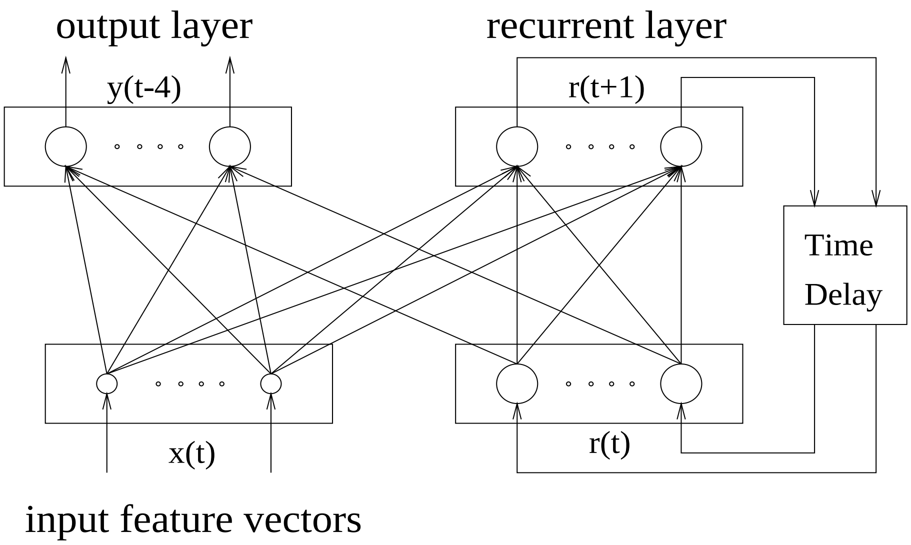

Figure 3.6: Cambridge recurrent neural network

The network is trained using backpropagation through time. This training method is computationally very expensive, researchers in Cambridge report training times of several days on a dedicated parallel computer. Also, due to potential instabilities inherent in a recurrent systems, training seems to require careful adjustment of learning parameters. The system has fewer parameters than a competitive mixture-of-Gaussian system which yields a faster decoding stage. Recently, the system was augmented to incorporate small neural networks to model context classes. This context-dependent system achieved the lowest reported error rate on the 1995 SQUALE continuous speech recognition evaluation 3.6.

3.5 Problems

All of today's existing hybrid speech recognition systems require special parallel hardware to be able to train the neural networks in a reasonable amount of time. Also, they require the choice of lots of parameters such as the learning rate, momentum factor or batch size. Although it was shown that large monolithic neural networks can do an excellent job in the computation of emission probabilities, they are mostly considered as 'black boxes'. Because of the lack of understanding how the networks perform the classification task, network weights are usually intialized with small random numbers which requires lots of iterations of backpropagation for the weights to converge. Mixtures of Gaussians based recognizers benefit from powerful initialization methods like k-means algorithm. Parameters for such systems usually converge within only 2-5 iterations of Forward-Backward training.

The major drawback of hybrid systems, however, is the inefficiency of gradient based training algorithms. Sizes of speech databases and neural networks in hybrid recognizers have gradually increased and will increase even further over the next years. Training times for such networks could become prohibitive, even with fast hardware.

Chapter 4

Hierarchical Mixtures of Experts

This chapter introduces Hierarchical Mixtures of Experts as a modular and hierarchical neural network for supervised learning. It closely follows the presentation by Jordan and Jacobs [27], yet focussing on classification instead of regression. The underlying statistical model will be discussed in detail, in order to motivate the presentation of an effective learning method for the architecture – the EM algorithm.

4.1 Introduction

The Hierarchical Mixture of Experts for the purpose of classification is a direct competitor to other, non-modular and non-hierarchical neural network classifiers such as the Multi Layer Perceptrons or the Radial Basis Function Networks, which have proven to be very powerful and general classifiers and function approximators. Therefore, the reader may ask questions like: Why do we need a modular, hierarchical network if we already have powerful methods for classification and regression? What are the drawbacks of traditional neural networks and other monolithic classifiers that lead to the development of modular and hierarchical architectures?

Fig. 4.1 shows a particular situation, where a modular approach to, in this case, function approximation yields significantly better results than traditional methods. The function to be approximated is piecewise linear with a discontinuity at $x = 0$. Clearly, the best way to approximate this kind of function is to split the task into two subregions, and apply standard linear regression to the data in each of the regions. This leads to the least possible number of parameters and the best approximation possible. The figure also shows a typical approximation obtained by an MLP or a higher order polynomial interpolation scheme. These methods usually produce smooth approximation surfaces not able to capture discontinuities like the one in our example. Even worse, the discontinuity leads to oscillations in the overall approximation surface that can only be reduced by using a larger number of parameters – which in turn leads to an unnecessarily increased model complexity.

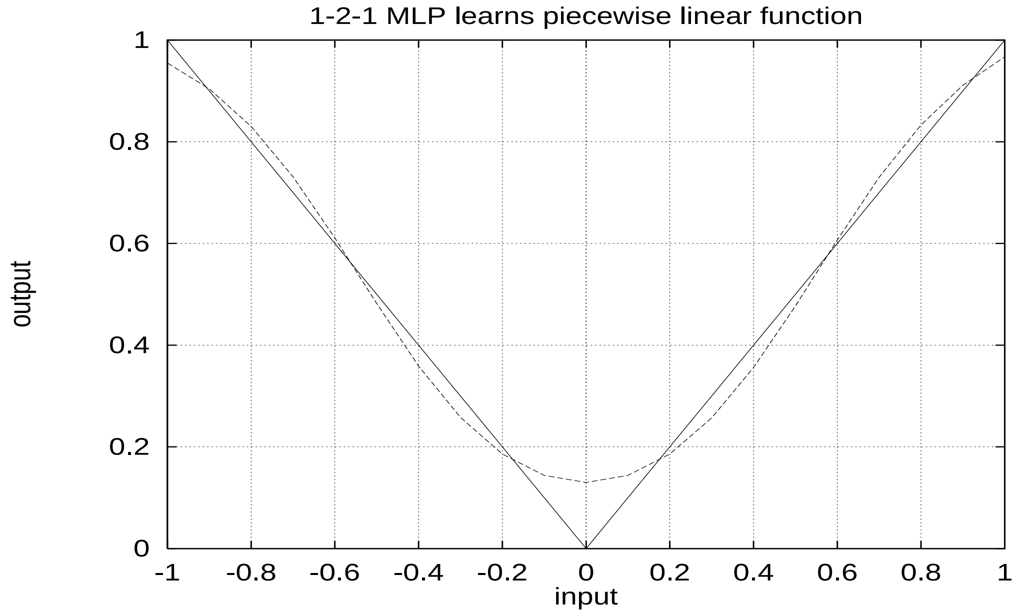

Figure 4.1: Learning to approximate a discontinuous function

Another major drawback of traditional neural networks is the complexity of their training algorithms, mostly based on gradient descent methods. This kind of training algorithm is slow and tedious, requiring the user to set various algorithmic parameters by trial and error. Training of large MLPs on very large databases (which is the case in hybrid speech recognition) requires such a large amount of CPU cycles, that even when using parallel implementations of backpropagation on dedicated hardware, researchers report training times of several days. This renders the analysis and optimization of learning parameters very time consuming, if at all possible.

It should be noted, that recent work in statistics has shown similarities between neural networks and statistical models such as generalized linear models, maximum redundancy analysis, projection pursuit and cluster analysis, that allow the application of much more efficient statistical learning/estimation techniques to the training of MLPs. In fact, it was shown, that an MLP with one hidden layer is essentially the same as the projection pursuit model, except that a MLP uses a predetermined functional form for the activation function in the hidden layer. Parameters of such a model can be estimated more efficiently by general purpose nonlinear modeling or optimization programs.

The remainder of this chapter will introduce a modular, hierarchical architecture for supervised learning that tackles all the discussed problems of standard neural networks.

4.1.1 Architecture

The Hierarchical Mixture of Experts architecture consists of relatively simple (i.e. one layer) gating and expert networks, organized in a tree structure as shown in Fig. 4.2. The basic principle behind this structure is the well known technique called divide-and-conquer. Algorithms of this kind solve complex problems by dividing it into simpler problems for which solutions can be obtained very easily. These partial solutions are then integrated to yield an overall solution to the whole problem. In the Hierarchical Mixtures of Experts architecture, the leaves of the tree represent expert networks, which act as simple local problem solvers. Their output is hierarchically combined by so called gating networks at the internal nodes of the tree to form the overall solution. To be more specific, the architecture has to learn an input-output mapping $\mathbf{y} = f(\mathbf{x})$ based on a set of training samples $\mathcal{T} = \{(\mathbf{x_i}, \mathbf{y_i}), i = 0, \ldots, N\}$. The expert networks as well as the gating networks receive the input vectors $\mathbf{x_i}$ with the difference that the gating networks use the input to compute confidence values for the outputs of their children, whereas the expert networks use the input to generate an estimate of the desired output value.

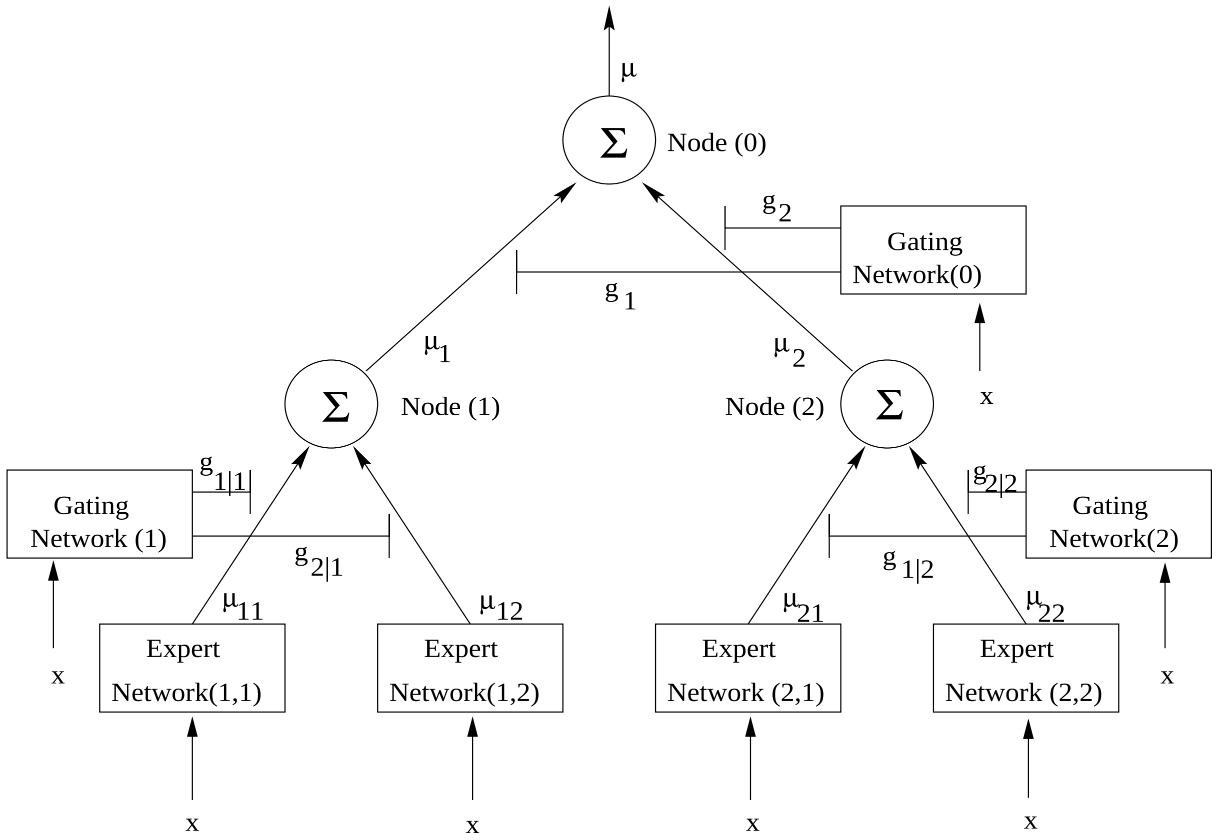

Figure 4.2: Hierarchical Mixtures of Experts Architecture

There are existing similar tree-structured divide-and-conquer models in statistics, namely CART by Breiman et. al. [6], MARS of Friedman [15] and ID3 by Quinlan [44]. However, these algorithms solve function approximation or classification problems by explicitly dividing the input space into subregions, such that only one single 'expert' is contributing to the overall output of the model. Caused by these 'hard-splits' of the input space, CART, MARS and ID3 tend to be variance-increasing, especially in the case of high-dimensional input spaces, where data is very sparsely distributed. In contrast, the gating networks in an HME are capable of computing soft splits of the input space, allowing input data to lie simultaneously in multiple regions. In this case, many experts contribute to the overall output which has a variance-decreasing effect.

All the expert networks in the HME tree realize linear mappings between the input and the output space with an additional output non-linearity. One can also interpret the experts as single layer perceptrons. In the case of multiway classification, the non-linearity is generally chosen to be the softmax function, whereas in the case of regression the non-linearity is the identity and the experts are strictly linear. The selection of the non-linearity depends on the probabilistic interpretation of the model and will be explained in the following section.

Consider the two-layer, binary branching HME in Fig. 4.2. Each of the expert networks (i, j) produces its output μ_{ij} from the input $\mathbf{x}$ according to:

$$\mu_{ij} = f(U_{ij}\mathbf{x})$$

where U_{ij} is a weight matrix and f is the output non-linearity. The input vector $\mathbf{x}$ is considered to have an additional constant coordinate value of 1.0 to allow for network biases.

The gating networks are also generalized linear. Since they perform multiway classification among the experts, the non-linearity is chosen to be the softmax non-linearity. The output values g_i of the top-level gating network are computed according to:

$$g_i = \frac{\exp(\xi_i)}{\sum_k \exp(\xi_k)} \qquad with \qquad \xi_i = \mathbf{v}_i^T \mathbf{x}$$

Due to the special form of the softmax non-linearity, the g_i are positive and sum up to one for each input vector $\mathbf{x}$. They can be interpreted as the local conditional probability, that an input vector $\mathbf{x}$ lies in the region of the associated children node. The lower level gating networks compute their output activations similar to the top-level gating network:

$$g_{j|i} = \frac{\exp(\xi_{ij})}{\sum_k \exp(\xi_{ik})} \qquad with \qquad \xi_{ij} = \mathbf{v}_{ij}^T \mathbf{x}$$

The output activations of the expert networks are weighted by the gating networks output activations as they proceed up the tree to form the overall output vector. Specifically, the output at the i-th internal node in the second layer of the tree is

$$\mu_i = \sum_j g_{j|i} \mu_{ij}$$

and the output at the top level (root node) is

$$\mu = \sum_i g_i \mu_i$$

Since both the expert and the gating networks compute their activations as a function of the input $\mathbf{x}$, the overall output of the architecture is a nonlinear function of the input (even in the case of linear experts). Furthermore, different input spaces may be used for gating and expert networks. In the case of speech recognition, the gating networks might be supplied with additional input features, e.g. speaking rate, in order to facilitate discrimination between different sounds.

4.1.2 Probabilistic Interpretation

The architecture is best understood as a generative probabilistic decision tree. Observable data is assumed to be generated by the model in the following way: For each input vector $\mathbf{x}$, the output values computed by the gating networks are interpreted as the multinomial probabilities of selecting one of the children nodes. Starting at the root node, a particular sequence of decisions is made based on the probability distributions imposed by the gating networks. This process eventually ends in a terminal node of the tree containing a specific expert network. This expert network computes a linear activation $\mathbf{mu}_{ij}$ using its weight matrix. The vector $\mathbf{mu}_{ij}$ is considered to be the mean of a probability density that models the generation of output vectors.

The gating networks parameterization corresponds to a multinomial logit probability model, which is a special case of a Generalized Linear Model (GLIM) [34]. That is, gating network outputs are assumed to follow a multinomial density

$$P(y_1, \ldots, y_n) = \frac{m!}{(y_1!) \ldots (y_n!)} p_1^{y_1} \ldots p_n^{y_n}$$

where the p_i are the multinomial probabilities associated with the different classes (in this case the children nodes) and $m = \sum_{i=1}^{n} y_i$ is generally taken to equal one for classification problems.

The probability density for the expert networks is assumed to be a member of the exponential family of densities. In the case of regression, the probabilistic component is generally chosen to be the Gaussian density

$$P(\mathbf{y}|\mathbf{x}, \theta) = \frac{1}{(2\pi)^{n/2} \prod_i \sigma_i} \exp\{-\frac{1}{2} \sum_i \frac{(x_i - \mu_i)^2}{\sigma_i^2}\}$$

whereas in the case of multiway classification, the expert's probability density function is the same as for the gating networks, with the difference, that the gating networks discriminate between children nodes and the expert networks discriminate between output classes.

Given these assumptions, the total probability of generating the output $\mathbf{y}$ from the input $\mathbf{x}$ can be given in form of a hierarchical mixture model:

$$P(\mathbf{y}|\mathbf{x},\theta) = \sum_i g_i(\mathbf{x},\mathbf{v_i}) \sum_j g_{j|i}(\mathbf{x},\mathbf{v}_{ij}) P(\mathbf{y}|\mathbf{x},\theta_{ij})$$

In this notation, θ contains both the gating network's parameters v_i, v_{ij} and the expert's parameters $\theta_{\mathbf{ij}}$.

4.1.3 Posterior Probabilities

In order to develop learning algorithms for the hierarchy, we need to introduce posterior node and posterior branch probabilities. Consider the training of a given HME architecture, where we explicitely know the desired output vector $\mathbf{y}$ for each input vector $\mathbf{x}$. In this context, we consider the gating probabilities g_i and $g_{j|i}$ to be *prior* branch probabilities, since they are computed based on the input vector $\mathbf{x}$ alone, without any knowledge about the target output vector $\mathbf{y}$. Using both the input and output vectors, *posterior* branch probabilities can be defined for the gating networks:

$$h_i = \frac{g_i \sum_j g_{j|i} P_{ij}(\mathbf{y})}{\sum_i g_i \sum_j g_{j|i} P_{ij}(\mathbf{y})} \qquad h_{j|i} = \frac{g_{j|i} P_{ij}(\mathbf{y})}{\sum_j g_{j|i} P_{ij}(\mathbf{y})}$$

Based on these conditional posterior probabilities, we can compute unconditional node probabilities for each node in the tree by multiplying all the conditional posterior branch probabilities along the path from the root node to the node in question. This way, we can assign a posterior probability to each of the expert networks too:

$$h_{ij} = h_i h_{j|i} = \frac{g_i g_{j|i} P_{ij}(\mathbf{y})}{\sum_i g_i \sum_j g_{j|i} P_{ij}(\mathbf{y})}$$

h_{ij} is interpreted as the probability that expert network (i,j) has generated the observed data pair $(\mathbf{x},\mathbf{y})$. Note, that posterior probabilities are not available during testing, where we do not have any knowledge about the target output vector $\mathbf{y}$. They are exclusively needed for the derivation of learning algorithms.

4.2 Gradient Ascent Learning

Since we assume that the HME realizes a probabilistic generative model of our data, we can define the likelihood of our model given a training set $\mathcal{T} = \{(\mathbf{x_i},\mathbf{y_i}), i = 0,\ldots,N\}$

and treat the learning problem as a maximum likelihood problem. This kind of learning algorithm for HMEs was introduced by Jordan and Jacobs [26]. The derivation of this learning algorithm is fairly straight forward and it can be realized both as an on-line and a batch learning method.

4.2.1 The Likelihood

It is common to use the log of the likelihood instead of the likelihood itself, which converts the product of probabilities to a sum:

$$\begin{aligned} l(\theta; \mathcal{X}) &= \sum_t \log P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta) \\ &= \sum_t \log \sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}}) \end{aligned}$$

In order to derive an update algorithm for the gating network and expert network parameters, we need the derivatives of the log likelihood with respect to the gating and expert parameters, respectively. For the top-level gating network, we obtain

$$\begin{aligned} \frac{\partial l(\theta; \mathcal{X})}{\partial v_k} &= \sum_t \frac{\sum_i (\partial g_i / \partial v_k) \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})}{\sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})} \\ &= \sum_t \frac{\sum_i g_i(\delta_{ik} - g_k) \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})}{\sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})} \mathbf{x}^{(t)} \\ &= \sum_t \frac{g_k \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}}) - g_k \sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})}{\sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})} \mathbf{x}^{(t)} \\ &= \sum_t (h_k - g_k) \mathbf{x}^{(t)} \end{aligned}$$

where we have used the derivative of the softmax function

$$\frac{\partial g_i}{\partial v_k} = g_i(\delta_{ik} - g_k)$$

Similarly, it can be shown that the derivative of the likelihood with respect to the second layer gating networks is

$$\frac{\partial l(\theta; \mathcal{X})}{\partial v_{kl}} = \sum_t h_k (h_{l|k} - g_{l|k}) \mathbf{x}^{(t)}$$

Since we are interested in the set of parameters that maximise the log likelihood of the observed data given the model, we perform gradient ascent in weight space using the likelihood gradients and a learning factor η to update the parameters of the gating networks:

$$
\begin{aligned}
v_i^{(k+1)} &= v_i^{(k)} + \eta \sum_t (h_i - g_i)\mathbf{x}^{(t)} \\
v_{ij}^{(k+1)} &= v_{ij}^{(k)} + \eta \sum_t h_i(h_{j|i} - g_{j|i})\mathbf{x}^{(t)}
\end{aligned}
$$

The above learning rule suggests an update after the presentation of the complete training set. Instead of computing the real gradients of the log likelihood over the whole training set, we could also use a variant, called stochastic gradient update, which updates the parameters each time a fixed number m of training samples have been presented to the architecture. This form of parameter update is usually called on-line learning and leads to faster convergence.

It remains to derive parameter update rules for the expert networks. Depending on the chosen probability density model for the expert networks, we obtain different update rules. Therefore we have to distinguish between regression and classification tasks and derive the different update algorithms in the next two sections.

4.2.2 Expert Parameter Updates for Regression

When the HME is used for function approximation, the underlying probability density is assumed to be Gaussian. To simplify the derivation of the update rule, we assume a unit variance Gaussian density, although update rules for Gaussians with full covariance matrices exist too. The gradient of the log likelihood with respect to the (k,l)-th expert is

$$
\begin{aligned}
\frac{\partial l(\theta; \mathcal{X})}{\partial \theta_{kl}} &= \sum_t \frac{g_k g_{l|k} (\partial P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{kl}})/\partial \theta_{kl})}{\sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})} \\
&= \sum_t h_{kl}(\mathbf{y}^{(t)} - \mu^{(t)})\mathbf{x}^{(t)T}
\end{aligned}
$$

which leads to the gradient update rule for expert parameters

$$
\theta_{ij}^{(k+1)} = \theta_{ij}^{(k)} + \eta \sum_t h_{ij}(\mathbf{y}^{(t)} - \mu^{(t)})\mathbf{x}^{(t)T}
$$

Note, that the above learning rule updates the whole weight matrix at once. If the hierarchy is capable of learning a given approximation problem perfectly, the differences between the target vectors $\mathbf{y}^{(t)}$ and the HME's linear predictions $\mu^{(t)}$ will eventually converge to zero. The gradient of the log likelihood will vanish and the updates will become zero.

4.2.3 Expert Parameter Updates for Classification

The objective of this thesis is to apply modular neural networks in a hybrid speech recognition environment. Therefore, we are mainly interested in the use of HMEs as classifiers and posterior class probability estimators. In the case of classification, the same kind of probability density applies to the expert and the gating networks, since they both perform multiway classification. However, for training a classifier, we usually have a data set with 'hard' targets. That means, there is a class label associated with each input vector $\mathbf{x}$. Using a 1-out-of-N encoding of class labels, the multinomial probability density degenerates as follows

$$P(t_1, \dots, t_n) = \frac{m!}{(t_1!)\dots(t_n!)} \mu_1^{t_1} \dots \mu_n^{t_n} = \mu_1^0 \dots \mu_c^1 \dots \mu_n^0 = \mu_c$$

Here, the μ_i are the output values of the classifier and the t_i are the target values for each class (which are zero for all but one class). μ_c stands for the output value associated with the correct target class. Using this simplified probability model, we obtain the derivative of the log likelihood with respect to the weight vector of node m in expert network (k, l)

$$\begin{aligned}
\frac{\partial l(\theta; \mathcal{X})}{\partial \theta_{klm}} &= \sum_t \frac{g_k g_{l|k} (\partial P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{kl}}) / \partial \theta_{klm})}{\sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})} \\
&= \sum_t \frac{g_k g_{l|k} (\partial \mu_{klc}^{(t)} / \partial \theta_{klm})}{\sum_i g_i \sum_j g_{j|i} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \theta_{\mathbf{ij}})} \\
&= \sum_t h_{kl} (\delta_{cm} - \mu_{klm}) \mathbf{x}^{(t)} \\
&= \sum_{t, c=m} h_{kl} (1 - \mu_{klm}) \mathbf{x}^{(t)} - \sum_{t, c \neq m} h_{kl} \mu_{klm} \mathbf{x}^{(t)}
\end{aligned}$$

which leads to the following expert network parameter update rule

$$\theta_{ijm}^{(k+1)} = \theta_{ijm}^{(k)} + \eta \Big(\sum_{t, c=m} h_{ij} (1 - \mu_{ijm}) \mathbf{x}^{(t)} - \sum_{t, c \neq m} h_{ij} \mu_{ijm} \mathbf{x}^{(t)} \Big)$$

Again, the update formulas can either be used in on-line or in batch mode. We will postpone the evaluation of the gradient ascent learning rule until after the next two sections, where we will derive a more efficient learning algorithm for the HME architecture.

4.3 EM Learning

The Expectation Maximization (EM) algorithm of Dempster et. al. [10] is a general technique for maximum likelihood estimation. It is mainly applied to unsupervised learning, i.e. clustering and mixture density estimation. The most popular application of EM to unsupervised learning in the context of speech recognition is the Baum-Welch or Forward-Backward algorithm that solves the learning problem for Hidden Markov Models. The EM algorithm is a very powerful iterative algorithm for maximum likelihod problems involving missing data. For example, in speech recognition, the Baum-Welch Reestimation usually converges in only 2-5 iterations. There is no reason, why the EM framework should not be applicable to supervised learning problems like the HME learning as well.

4.3.1 General EM Algorithm

The iterative EM algorithm is composed of two steps. The E-step (Expectation) defines a new likelihood function in each iteration, that is maximised during the M-step (Maximization). Often, E- and M-step are combined in a single undivisible algorithm, but for theoretical purposes we will distinguish between the two steps. If the M-step only increases the likelihood instead of maximizing it in each step, the algorithm is called Generalized Expectation Maximization (GEM). The learning algorithm for the Boltzmann machine, for example, is essentially a GEM algorithm.

In order to apply the EM algorithm to a new domain, a set of 'missing' or 'unknown' variables have to be defined, that would simplify the optimization of the log likelihood, if they were known. We then distinguish between the *incomplete-data* log likelihood $l(\theta; \mathbf{X})$ over the observable data $\mathbf{X}$ and the *complete-data* log likelihood $l_c(\theta; \mathbf{Y})$ over the extended data $\mathbf{Y} = \mathbf{X} \cup \mathbf{Z}$ which includes the set of missing variables $\mathbf{Z}$. It is important to note, that the complete-data log likelihood is a random variable because the set of missing variables are unknown.

The EM algorithm aims at increasing an estimation of the complete-data log likelihood as follows. Using the observed data and the current model, the E-step first computes the expected value of the complete-data log likelihood:

$$Q(\theta, \theta^{(k)}) = E[l_c(\theta; \mathcal{Y})|\mathcal{X}]$$

The superscript k refers to the parameters at the k-th iteration of the algorithm. The E-step yields a deterministic function Q of the parameters of the model. The M-step maximizes the Q-function with respect to the model's parameters:

$$\theta^{(k+1)} = \arg\max_{\theta} Q(\theta, \theta^{(k)}$$

The process iterates by looping over E- and M-step until the maximization yields no further improvement. The EM algorithm guarantees to compute parameter estimates that increase the Q-function in each iteration. The Q-function, however, is just the expected value of the complete-data log likelihood. Our goal is to maximize the incomplete-data log likelihood. Dempster et. al. addressed this issue and proved that an increase in the Q-function always implies an increase in the incomplete-data log likelihood:

$$Q(\theta, \theta^{(k+1)}) \geq Q(\theta, \theta^{(k)}) \qquad \Longrightarrow \qquad l(\theta^{(k+1)}; \mathcal{X}) \geq l(\theta^{(k)}; \mathcal{X})$$

That means, the original likelihood $l(\theta; \mathcal{X})$ increases monotonically with every iteration, converging to a local minimum.

4.3.2 Applying EM to the HME

Application of EM to the HME architecture involves the definition of 'missing' variables that facilitate the optimization of the log likelihood. Let $z_i, i = 1, \ldots, n$ be a set of binary indicator variables for the top-level gating network, and let $z_{j|i}, i, j = 1, \ldots, n$ be a set of binary indicator variables for the second layer gating networks. For any given input vector $\mathbf{x}$ exactly one of the z_is is one, all the others are zero. Similarily, given the z_i, exactly one of the $z_{j|i}$ is one, all the others are zero. The z_is and $z_{j|i}$s have an interpretation as the (unknown) decisions corresponding to the probability model. An instantiation of the z_is and $z_{j|i}$s corresponds to a specific path from the root node of the tree to one of the leaves, determining the expert responsible for data generation. Note, however, that the z_is and $z_{j|i}$s are not known and must be treated as random variables. If they were known, the maximum likelihood problem for the HME would decouple into a set of independent maximum likelihood problems for each of the gating and expert networks. Although the z_is and $z_{j|i}$s are unknown, we can specify a complete-data log likelihood probability model that links them to the observable data and allows for the application of the EM algorithm:

$$\begin{aligned}
l_c(\theta; \mathcal{Y}) &= \log \prod_t \prod_i \prod_j g_i^{(t)} g_{j|i}^{(t)} P_{ij}(\mathbf{y}^{(t)})^{z_{ij}^{(t)}} \\
&= \sum_t \sum_i \sum_j z_{ij}^{(t)} \log\{g_i^{(t)} g_{j|i}^{(t)} P_{ij}(\mathbf{y}^{(t)})\} \\
&= \sum_t \sum_i \sum_j z_{ij}^{(t)} \{\log g_i^{(t)} + \log g_{j|i}^{(t)} + \log P_{ij}(\mathbf{y}^{(t)})\}
\end{aligned}$$

The above complete-data log likelihood is much easier to maximize than the corresponding incomplete-data log likelihood, because we managed to bring the logarithm inside the summation.

One can prove easily that the posterior probabilities h_i, $h_{j|i}$ and h_{ij} can be used as the expected values for the unknown indicator variables z_i, $z_{j|i}$ and z_{ij}, respectively (see [26] for a proof). Using this fact, we can define the Q-function for the E-step of the EM algorithm:

$$Q(\theta, \theta^{(k)}) = \sum_t \sum_i \sum_j h_{ij}^{(t)} \{\log g_i^{(t)} + \log g_{j|i}^{(t)} + \log P_{ij}(\mathbf{y}^{(t)})\}$$

The M-step requires the maximization of the Q-function with respect to the model's parameters. We now see the benefits of applying EM, since the maximization decouples into a set of separate maximum likelihood problems that may be solved independently during the M-step:

$$
\begin{aligned}
\mathbf{v}_i^{(k+1)} &= \arg\max_{\mathbf{v}_i} \sum_t \sum_l h_l^{(t)} \log g_l^{(t)} \\
\mathbf{v}_{j|i}^{(k+1)} &= \arg\max_{\mathbf{v}_{j|i}} \sum_t \sum_l h_l^{(t)} \sum_m h_{m|l}^{(t)} \log g_{m|l}^{(t)} \\
\theta_{ij}^{(k+1)} &= \arg\max_{\theta_{ij}} \sum_t h_{ij}^{(t)} \log P_{ij}(\mathbf{y}^{(t)})
\end{aligned}
$$

Since we are mainly interested in the HME as a classifier, we will restrict the derivation of solutions for the above maximum likelihood problems to this case, assuming a multinomial (Poisson) density as the probabilitiy model for the expert as well as the gating networks. Under these assumptions, the log likelihood equation for the expert and gating network's parameters are weighted log likelihoods for a special case of a Generalized Linear Model (GLIM), namely a *multinomial logit* model. For the top-level gating networks, we have to maximize the cross-entropy between the posterior branching probabilities h_l and the branching (prior) probabilities g_l. For the second level gating networks, we have to maximize the cross-entropy between the posterior branching probabilities $h_{m|l}$ and the branching (prior) probabilities $g_{m|l}$, weighted by the posterior probability h_l of the gating node itself. In deeper trees, the weight for the cross-entropy is simply the product of posterior branching probabilities along the path from the root node to the gating node in question. Finally, the maximization problem for the expert networks involves maximizing the cross-entropy between the expert's posterior probability and the output at the node of the actual correct class. Since all of the above maximization problems are based on likelihoods for generalized linear models, we can apply an algorithm called Iteratively Reweighted Least Squares (IRLS) [34] that solves such likelihood problems.

4.3.3 Iteratively Reweighted Least Squares (IRLS)

Applying the EM algorithm to the HME architecture requires the computation of posterior probabilities h_i, $h_{j|i}$ and h_{ij} for each input vector $\mathbf{x}$ in the E-step, and the maximization of independent maximum likelihood problems for GLIMs in the M-step. This process is iteratively repeated until no further improvement can be obtained. This section describes the IRLS algorithm that can be used to solve the maximization problems within the M-step. The IRLS algorithm is a special case of the Fisher scoring method [12]. In order to maximize the log likelihood $l(\beta; \mathcal{X})$ with respect to the parameter vector β, the Fisher scoring method updates β according to

$$\beta^{(k+1)} = \beta^{(k)} - \left\{ E\left[\frac{\partial l(\beta^{(k)};\mathcal{X})}{\partial\beta\partial\beta^T} \right] \right\}^{-1} \frac{\partial l(\beta^{(k)};\mathcal{X})}{\partial\beta}$$

This equation strongly resembles the Newton-Raphson equation with the notable difference that in the Fisher scoring method, the Hessian is replaced by the expected value of the Hessian. Besides the fact, that the expected value of the Hessian is often easier to compute, there are statistical reasons for prefering it over the actual Hessian.

We will now derive the IRLS algorithm for the special case of a multinomial GLIM. The multinomial density is a member of the exponential families of distributions which is an important class of distributions in statistics. It can be rewritten in the following form:

$$\begin{aligned}
P(y_1,\ldots,y_n) &= \frac{m!}{(y_1!)\ldots(y_n!)} p_1^{y_1}\ldots p_n^{y_n} \\
&= \exp\left\{ \log\frac{m!}{(y_1!)\ldots(y_n!)} + \sum_{i=1}^{n} y_i \log p_i \right\} \\
&= \exp\left\{ \log\frac{m!}{(y_1!)\ldots(y_n!)} + \sum_{i=1}^{n-1} y_i \log\frac{p_i}{p_n} + n\log p_n \right\}
\end{aligned}$$

where we have used the constraint that the p_i sum up to one to express p_n as $p_n = 1 - \sum_{i=1}^{n-1} p_i$. Comparing this form of the multinomial density with the general form of a density of the exponential family

$$P(\mathbf{y},\eta,\Phi) = \exp\left\{ \frac{(\eta\mathbf{y} - b(\eta))}{\Phi} + c(\mathbf{y},\Phi) \right\}$$

with the *natural* parameter η and the *dispersion* parameter Φ, we can define the natural parameter η to be the vector of η_is:

$$\begin{aligned}
\eta_i &= \log\frac{p_i}{p_n} \\
&= \log\frac{p_i}{1-\sum_{i=1}^{n-1} p_i} \\
&= \log\left\{ p_i \left(1 + \sum_{j=1}^{n-1} \exp(\eta_j) \right) \right\}
\end{aligned}$$

This equation can be inverted to yield

$$\begin{aligned}
p_i &= \frac{\exp(\eta_i)}{1+\sum_{j=1}^{n-1}\exp(\eta_j)} \\
&= \frac{\exp(\eta_i)}{\sum_{j=1}^{n}\exp(\eta_j)}
\end{aligned}$$

which is the 'softmax' function that we have assumed as the non-linearity for the gating and expert networks. By parameterizing the multinomial probability density in terms of the natural parameter η, we have forced the choice of the network's output non-linearity to be the softmax function. The softmax function is refered to as the *canonical link* to the multinomial distribution. Other choices of the output probability density result in different canonical links, for example, assuming a Bernoulli density yields the standard sigmoid function as the canonical link function.

Having justified the choice of the output non-linearity, we now proceed in the derivation of the IRLS update equations. First we define the function b implicit as the integral of the softmax function:

$$\mu_i^{(t)} = \frac{\partial b}{\partial \eta_i} = \frac{\exp(\eta_i^{(t)})}{\sum_j \exp(\eta_j^{(t)})} \qquad \text{with} \qquad \eta_i^{(t)} = \beta_{\mathbf{i}}{}^T \mathbf{x}^{(t)}$$

We can now compute the terms necessary for the Fisher scoring equation, that is, we need the likelihood and the first and second derivatives of the likelihood of a multinomial GLIM:

$$\begin{aligned}
l(\beta; \mathcal{X}) &= \sum_t \sum_k \left(\beta_{\mathbf{k}}{}^T \mathbf{x}^{(t)} y_k^{(t)} - b(\beta_k^T \mathbf{x}^{(t)}) \right) + \log \frac{m!}{(y_1!) \dots (y_n!)} \\
\frac{\partial l(\beta; \mathcal{X})}{\partial \beta_i} &= \sum_t \sum_k \left(\delta_{ki} y_k^{(t)} - \frac{\partial b(\beta_k^T \mathbf{x}^{(t)})}{\partial \beta_i} \right) \mathbf{x}^{(t)} \\
\frac{\partial l(\beta; \mathcal{X})}{\partial \beta_i \partial \beta_j^T} &= \sum_t \sum_k \frac{\partial b(\beta_k^T \mathbf{x}^{(t)})}{\partial \beta_i \partial \beta_j^T} \mathbf{x}^{(t)} \mathbf{x}^{(t)T}
\end{aligned}$$

Finally, by assembling all these equations into the Fisher scoring update function, we obtain the following IRLS algorithm for multinomial GLIMs:

$$\beta_i^{(k+1)} = \beta_i^{(k)} + \left(X^T W_i X \right)^{-1} X^T W_i \mathbf{e}_i$$

where W_i is a diagonal matrix with diagonal elements

$$w_i^{(t)} = \sum_k \left[\mu_k^{(t)} (\delta_{ki} - \mu_i^{(t)}) \right]$$

and $\mathbf{e}_i$ is the vector of scalars $e_i^{(t)}$:

$$e_i^{(t)} = y_i^{(t)} - \mu_i^{(t)}$$

The weight matrices W_i and the vectors $\mathbf{e_i}$ change from iteration to iteration because they depend on the weight vectors β_i. The above update equation is essentially a solution to a weighted least squares problem. In our case, we need to extend the IRLS algorithm because we have additional fixed observation weights imposed by the gating networks. This can easily be done by multiplying the fixed observation weights with the iteratively varying weight matrices W_i, which leads to an iteratively reweighted *weighted* least squares algorithm. Applying this algorithm to the HME architecture yields the following training method:

1. **Expectation Step:**
   Compute posterior branching/node probabilities $h_i^{(t)}$, $h_{j|i}^{(t)}$ and $h_{ij}^{(t)}$ for each data pair $(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})$ of the training set.

2. **Maximization Step:**

   (a) **Inner loop for experts:**
   For each expert network, solve an IRLS problem with observations $(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})$ and observation weights $h_{ij}^{(t)}$.

   (b) **Inner loop for top-level gates:**
   For each top-level gating network, solve an IRLS problem with observations $(\mathbf{x}^{(t)}, h_i^{(t)})$.

   (c) **Inner loop for second-level gates:**
   For each second-level gating network, solve a weighted IRLS problem with observations $(\mathbf{x}^{(t)}, h_{j|i}^{(t)})$ and observation weights $h_i^{(t)}$.

3. Iterate EM steps using the updated parameter values.

This EM algorithm, though being quite effective, needs an iterative procedure in the M-step, while posterior probabilities need to be stored temporarily. This is not feasible when dealing with large data sets, as is the case in speech recognition. Therefore, we are interested in a version of the EM algorithm, that allows to solve the maximization steps in one pass. There are two ways of achieving this. The first one is, to relax the constraint of maximization in the M-step and derive a Generalized EM algorithm (GEM) that only guarantees to increase the log-likelihoods during the M-step. The other way is to use least squares fitting instead of likelihood maximization together with heuristics to derive a practically useful learning algorithm, which we will do in the next section.

4.4 Least Squares and Heuristics

Recall the three maximization problems derived from the Q-function and which we want to solve in a one-pass algorithm:

$$\begin{aligned}
\mathbf{v}_i^{(k+1)} &= \arg\max_{\mathbf{v}_i} \sum_t \sum_l h_l^{(t)} \log g_l^{(t)} \\
\mathbf{v}_{j|i}^{(k+1)} &= \arg\max_{\mathbf{v}_{j|i}} \sum_t \sum_l h_l^{(t)} \sum_m h_{m|l}^{(t)} \log g_{m|l}^{(t)} \\
\theta_{ij}^{(k+1)} &= \arg\max_{\theta_{ij}} \sum_t h_{ij}^{(t)} \log P_{ij}(\mathbf{y}^{(t)})
\end{aligned}$$

Computing the derivatives of the log likelihoods with respect to the parameters $\mathbf{v}_i$, $\mathbf{v}_{j|i}$ and θ_{ijk} respectively, and setting them to zero yields:

$$\begin{aligned}
\sum_t \left\{ h_i^{(t)} - g_i^{(t)} \right\} \mathbf{x}^{(t)} &= 0 \\
\sum_t \left\{ h_i^{(t)} \left(h_{j|i}^{(t)} - g_{j|i}^{(t)} \right) \right\} \mathbf{x}^{(t)} &= 0 \\
\sum_t \left\{ h_{ij}^{(t)} \left(t_k^{(t)} - \mu_{ijk}^{(t)} \right) \right\} \mathbf{x}^{(t)} &= 0
\end{aligned}$$

In the above equations, one can think of the posteriors as being targets for the gating and expert network outputs. As mentioned before, the posteriors are estimates of the unknown indicator variables which would be the correct targets, if they were known. By inverting the softmax non-linearity at the outputs of gating and expert networks, we can compute targets for the linear predictors which, in turn, can be used for standard least squares fitting. Inverting the softmax function

$$y_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \qquad \text{yields} \qquad x_i = \log y_i + \log \sum_j \exp(x_j) = \log y_i + C$$

The second term is constant for all x_i and constant terms common to all x_is disappear when the softmax function is applied. Therefore, we can use the $\log y_i$s as targets for the linear predictors. In the case of the gating networks we obtain the following one-pass least squares solutions to the maximization problem:

$$\begin{aligned}
\mathbf{v}_i &= (X^T X)^{-1} X \mathbf{e} \\
\mathbf{v}_{j|i} &= (X^T W X)^{-1} X W \mathbf{f}
\end{aligned}$$

with $\mathbf{e} = (\log h_i^{(1)}, \ldots, \log h_i^{(N)})$, $\mathbf{f} = (\log h_{j|i}^{(1)}, \ldots, \log h_{j|i}^{(N)})$, $W = I(h_i^{(1)}, \ldots, h_i^{(N)})$.

However, trying to compute targets for the linear predictors of the expert networks, we face the problem of having to compute the log of zero since all but one coefficient of the target vectors are zero. The heuristic here is, to use targets t_i out of $\{\epsilon, 1\}$ instead of the usual $\{0,1\}$. In practice, the value of ϵ is subject to optimization, but small values around $1e-3$ have proven to work well. Thus, the least squares problem for expert networks is solvable as before:

$$\theta_{ijk} = (X^T W X)^{-1} X W \mathbf{e}$$

with $\mathbf{e} = (\log t_k^{(1)}, \ldots, \log t_k^{(N)})$ and $W = I(h_{ij}^{(1)}, \ldots, h_{ij}^{(N)})$. Using standard (weighted) least squares, we were able to derive an effective EM algorithm with a one-pass M-step, suitable for large hierarchies and large data sets. During training, we have to compute posterior probabilities and accumulate the weighted input vectors into the least squares matrices and vectors. After one iteration, a single matrix inversion for each expert/gating network and a matrix-vector multiplication yields new parameter estimates. In the reminder of this chapter, we will evaluate the EM algorithm and the gradient ascent algorithm in terms of accuracy, generalization and convergence speed on a relatively small task. We will also compare the HME with a multi layer perceptron (MLP) trained by error-backpropagation. The integration of HME's into a hybrid speech recognition framework will be evaluated later in a separate chapter.

4.5 HME for Vowel Classification

We will demonstrate the properties of the HME architecture and its learning algorithms on Peterson and Barneys vowel classification data set [42]. We chose this dataset because it is non-artificial, speech recognition related and relatively small, allowing to explore and analyze the space of learning parameters. Another advantage of this dataset is its low dimensionality. We can easily reduce the originally four-dimensional feature vectors to two-dimensional feature vectors, which allows us to draw certain properties of the classfiers in a two dimensional coordinate system. We think that this kind of analysis provides deeper insight and better understanding of the way, the HME works.

4.5.1 The Data Set

The data set consists of 1520 four dimensional feature vectors. The feature coefficients are the formant frequencies F0,F1,F2 and F3. The data set contains an equal number of training vectors for each of the following 10 American English vowels (uniform prior distribution).

IY IH EH AE AH AA AO UH UW ER

We did not preprocess the data in any way, except that we normalized each of the four formant frequencies in the data set independently to the range $[0,1]$. Fig. 4.3 shows the complete data set in the normalized (F1,F2) feature space.

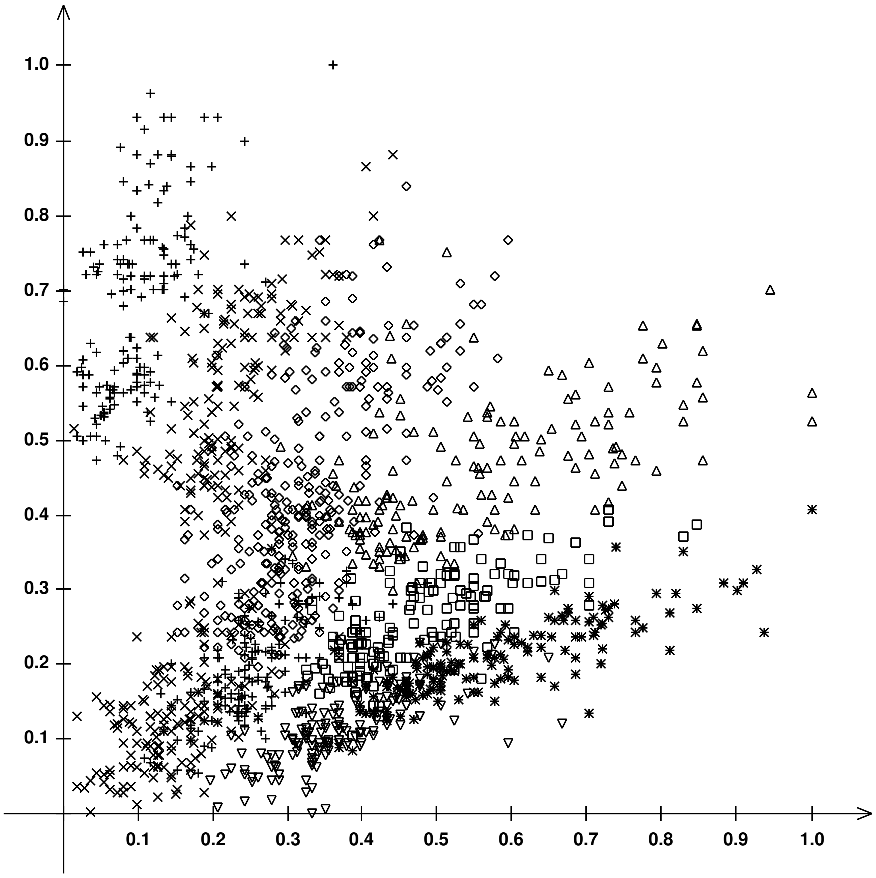

Figure 4.3: Peterson & Barneys vowel classification data set x =F1, y =F2

Syrdal and Gopal [50] performed classification on this dataset using a quantitative perceptual model of human vowel recognition. They reported classification rates between 82.3% and 85.9% for their classifier based on bark scale differences and linear discriminant analysis (LDA). Human listeners achieved an average classification rate of 94.4% when hearing the original recordings of the vowels.

4.5.2 Results

Fig. 4.4 shows the evolution of the likelihood on the training data and the mean square error and the classification error on the test data for a GLIM a MLP and different HME architectures (branching factor 2, depth 1,2 and 3). The HME's were trained with a combination of the Least Squares heuristic to EM and the gradient ascent algorithm. We found, that the Least Squares heuristic converges very fast (faster than the gradient based training) but is not able to achieve the same performance.

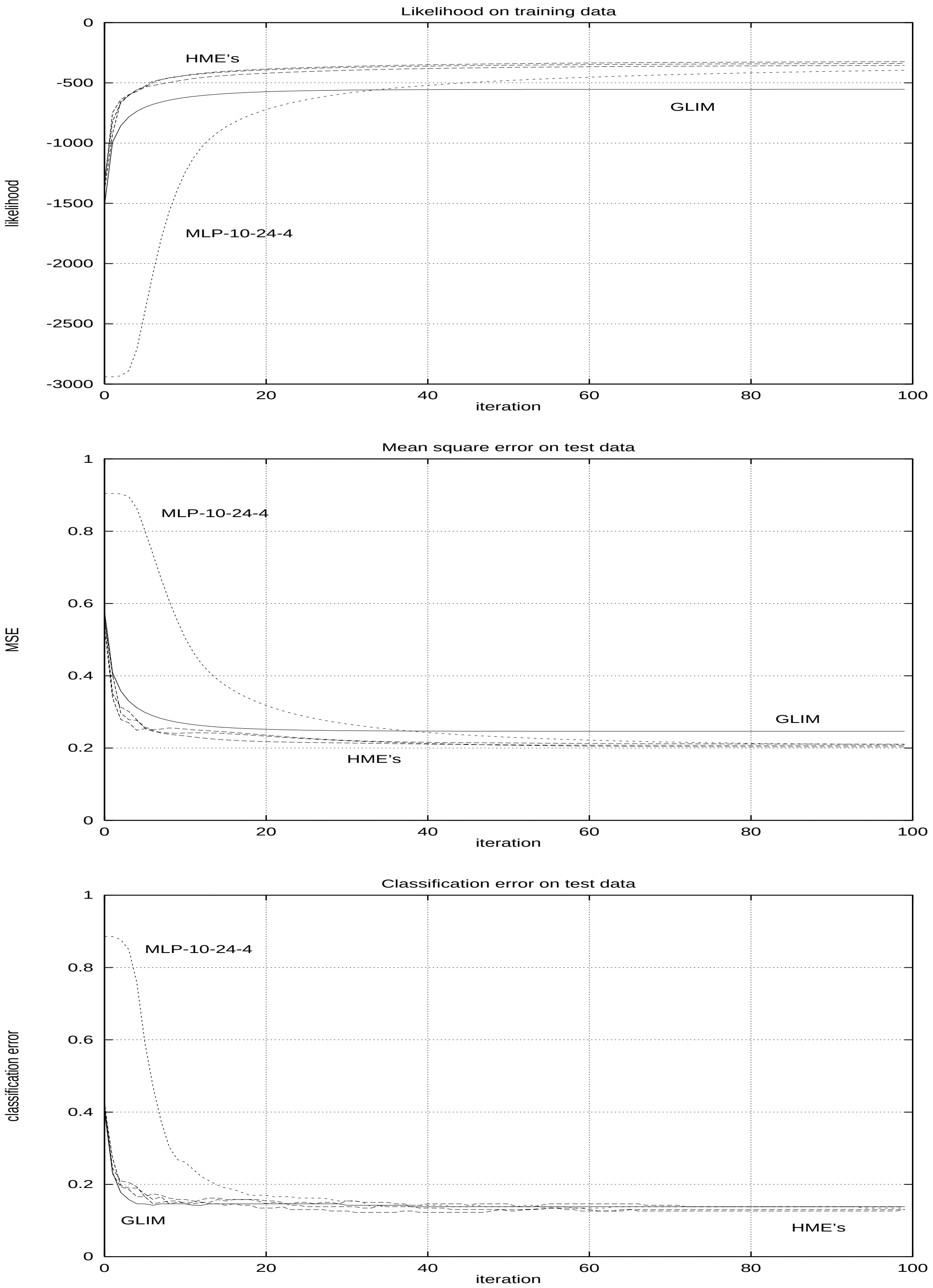

Figure 4.4: Typical training runs on Peterson&Barneys vowel data

Therefore, we used LS for the first few iterations before switching to GA, which gave the best results. The MLP was trained with on-line stochastic gradient error backpropagation with a learning rate of 0.1 (optimized by several trials). The training runs were performed on 4-dimensional feature vectors. Comparing classifier performances with respect to the classification error rate, one can see that a simple GLIM is competitive with both a 2-layer MLP with 24 hidden units and the different HME architectures. However, the evolution of the likelihood and mean square error show that MLP and HME's are able to learn the data better. Several things deserve to be mentioned:

- MLP and HME's achieve roughly the same performance
- Convergence is much faster for the HME's due to the EM algorithm
- Different HME architectures do not vary significantly in the case of the vowel data.

Fig. 4.5 shows the class boundaries imposed on a 2-dimensional feature space (F1,F2) by an HME (depth 3,branching factor 2) and an MLP (24 hidden units), respectively.

Figure 4.5: Class boundaries obtained by HME (left) and MLP (right)

HME and MLP were trained until convergence on the 2-dimensional feature. The plots in Fig. 4.5 were computed by sampling the interval $[0, 1]^2$, coloring the class with highest output activation in different shades of gray. The MLP seems to prefer non-linear curvy class boundaries, whereas the HME imposes almost linear ones. It seems that the HME discovers that the task does not need a soft collaboration between experts, therefore partitioning the input space into disjunct segments, which are classifier by the (generalized) linear experts.

Fig.4.6 shows the evolution of the activation regions of the experts while training the architecture. The plots are sampled in the same region $[0,1]^2$ as before, coloring the expert with the highest cummulative gating probability in different shades of gray. Obviously, as the training proceeds, the HME shuts off 5 of its 8 experts completely. A combination of 3 experts seems to be enough to solve the given task. This again means, that a lot of parameters in the HME tree are rendered useless in this specific application.

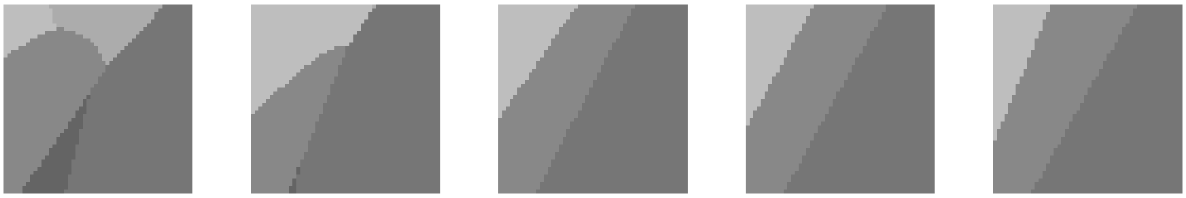

Figure 4.6: Evolution of expert's regions of activation (after 1,2,3,4 and 9 iterations, respectively)

Since we do not know in advance, how many experts are sufficient to solve a given problem adequately, we can only guess and use an architecture that is likely to contain more experts than needed. This approach to model selection is clearly a waste of parameters. The next chapter addresses this problem by presenting a constructive method which iteratively grows an HME architecture that uses its parameters more effectively.

Chapter 5

Constructive Methods

5.1 Motivation

One of the essential problems with the HME approach, as with other neural architectures, is model selection. Applying HME's to a classification or regression problem requires the choice of structural parameters such as the tree depth and the branching factor. As with other architectures, the problem of model selection is mostly solved in a rather simple way. Architectures of different size are trained and their performances are compared on an independent test set to select the one, that generalizes best. This approach is computationally very expensive especially when dealing with large data sets.

Better solutions to selecting model sizes are constructive and/or pruning methods. Constructive methods iteratively generate larger models starting from a very small one. For example, Fahlman's cascade correlation algorithm realizes such a constructive method for a special multi-layered network. The basic idea in all growing algorithms is to use some criterion on the training data to select the locally best expansion out of the set of all possible expansions to adaptively generate an architecture that fits the data better than its static counterpart.

Pruning methods, on the other hand, use the opposite strategy: A large (possibly oversized) architecture is evaluated to detect obsolete or ineffective parts which then are removed before the architecture is re-trained. This process can also be repeated iteratively using the performance on an independent test set as the stopping criterion. Computationally, pruning methods have the disadvantage of repeatedly requiring the training of unnecessarily large architectures.

Because of the inherent tree structure of the HME, it is very appealing to derive a growing algorithm for this architecture. The machine learning literature offers a wide variety of growing algorithms for classification and decision trees [44], [45], [6]. Unfortunately, these algorithms require the evaluation of the gain of all possible node splits, using (mostly) entropy or likelihood based criterions, to eventually realize the best split and discard all the others. Waterhouse and Robinson [56] presented such an algorithm for the HME architecture. They evaluated their growing algorithm on a relatively small data set. In the case of very large speech data sets, their approach is no longer applicable in a reasonable amount of time. We therefore developed a different growing algorithm for the HME architecture which imposes very little overhead and which is applicable in our domain.

5.2 Algorithms

We distinguish between tree growing and tree pruning, although both techniques are usually applied simultaneously, in order to achieve faster learning and recognition passes.

5.2.1 Adaptive Tree Growing

In order to grow an HME, we have to define an evaluation criterion to score the experts performance on the training data, which in turn will allow us to select the worst expert to be split into a new subtree, providing additional parameters which can help to overcome the errors made by this expert.

Viewing the HME as a probabilistic model of the observed data, we partition the input dependent likelihood of data generation using the expert selection probabilities provided by the gating networks

$$\begin{aligned} l(\mathbf{\Theta};\mathcal{X}) &= \sum_t \log P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)},\mathbf{\Theta}) = \sum_t \sum_k g_k \log P_k(\mathbf{y}^{(t)}|\mathbf{x}^{(t)},\mathbf{\Theta}_k) \\ &= \sum_k \sum_t \log[P_k(\mathbf{y}^{(t)}|\mathbf{x}^{(t)},\mathbf{\Theta_k})]^{g_k} = \sum_k l_k(\mathbf{\Theta_k};\mathcal{X}) \end{aligned}$$

where the g_k are the products of the gating probabilities along the path from the root node to the k-th expert, that is, g_k is the probability that expert k is responsible for generating the observed data (note, that the g_k sum up to one). The expert-dependent scaled likelihoods $l_k(\mathbf{\Theta};\mathcal{X})$ can be used as a measure for the performance of an expert within its region of responsibility. We use this measure as the basis of our tree growing algorithm:

1. Initialize and train a simple HME consisting of only one gate and several experts.

2. Compute the expert-dependent scaled likelihoods $l_k(\mathbf{\Theta};\mathcal{X})$ for each expert in one additional pass through the training data.

3. Find the expert k with minimum l_k and expand the tree, replacing the expert by a new gate with random weights and new experts that copy the weights from the old expert with additional small random perturberations.

4. Train the architecture to a local minimum of the classification error using a cross-validation set.

5. Continue with step (2) until desired tree size is reached.

The number of tree growing phases may either be pre-determined, or based on difference in the likelihoods before and after splitting a node. In contrast to the growing algorithm in [56], our algorithm does not hypothesize all possible node splits, but determines the expansion node(s) directly, which is much faster, especially when dealing with large hierarchies.

5.2.2 Pruning

Furthermore, we implemented a path pruning technique similar to the one proposed in [56], which speeds up training and testing times significantly. During the recursive depth-first traversal of the tree (needed for forward evaluation, posterior probability computation and accumulation of node statistics) a path is pruned temporarily if the current node's probability of activation falls below a certain threshold. Additionally, we also prune subtrees permanently, if the sum of a node's activation probabilities over the whole training set falls below a certain threshold. This technique is consistent with the growing algorithm and helps prevent instabilities and singularities in the parameter updates, since nodes that accumulate too little training information will be pruned away, without being considered for a parameter update.

Temporarily pruning branches of the HME tree can speed up training and testing times considerably, although this will most likely lead to an increase in error rate. We will present results of experiments with different pruning thresholds and their impact on the performance of an HME system. For speech recognition applications, a means for trading off accuracy against speed is very appealing, especially for demo systems, where the system's reaction time is more important than its performance (although an improvement in both directions is desirable, of course). We will therefore also examine the effect of HME pruning on speech recognition performance.

5.3 Experiments

We evaluate the tree growing and pruning algorithms on the Peterson & Barney vowel classification task, comparing the resulting HME's with standard pre-determined HME architectures.

5.3.1 Tree Growing

We compare a standard binary tree HME (depth 3) containing 8 experts with an adaptively grown binary HME with the same number of experts. Fig. 5.1 and Fig. 5.2 show the evolution of the classification rate and log-likelihood during training. The standard HME achieves it's final performance after 9 iterations, the growing HME is able to achieve the same performance after 8 iterations, at this time consisting of only 3 experts. This is consistent with our earlier observations.

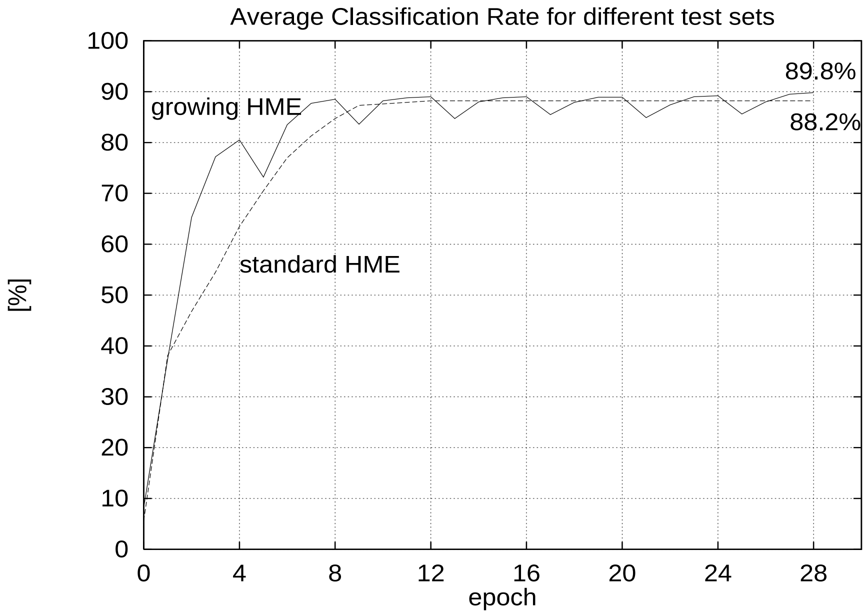

Figure 5.1: Classification rate for standard and growing HME

The bumpiness of the curves for the growing HME are due to the node splitting, that was done after every 4 iterations. Each time a node is being split, two new experts are introduced and initalized by the splitting candidate's parameters with small additional random perturberations. This causes an initial decrease in both classification rate and log-likelihood which is soon redeemed by the power of additional parameters.

One of the motivations for the growing algorithm was the desire to use the available parameters effectively. Fig. 5.3 and Fig. 5.4 compare the two architectures in this respect. They show the final topologies together with histograms at each internal node, approximating the distributions of gating probabilities over the test set. The histogram trees should be interpreted as follows:

- A sharp peak at the left or right side of a histogram indicates that one of the two children nodes is shut off by the corresponding gate.

- Peaks both at the left and the right side of a histogram indicate a more or less hard split of the input space by the corresponding gate.

- A peak in the middle of the histogram indicates that the corresponding gate makes use of soft splits of the input space.

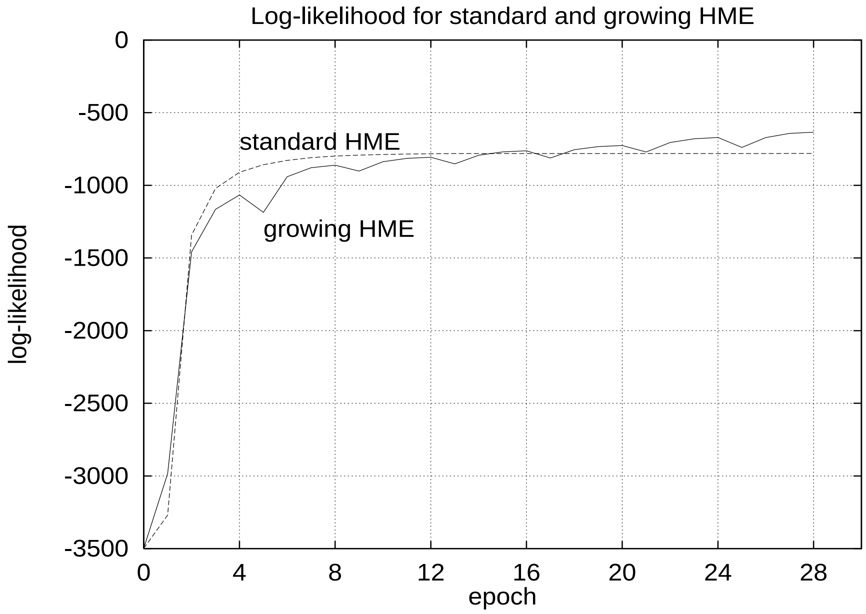

Figure 5.2: Log-likelihood for standard and growing HME

As one can see in Fig. 5.3, only 4 of the 8 experts can contribute to the overall output of the hierarchy, the remaining 4 experts are 'pinched-off' almost completely.

Fig. 5.4 shows the same histogram tree for the grown architecture. Here, almost all experts contribute to the overall output. The criterion for splitting nodes during the growing phase implicitely guarantees this because the splitting score is weighted by the experts activation. An expert that is hardly ever active will never be split into a new subtree which is exactly what we want.

Fig. 5.5 and Fig. 5.6 compare the regions of activation for each of the 8 experts in both architectures. Each plot was obtained by sampling the expert's activation (product of gating probabilities along the path from root to expert node) in the region $[0,1]^2$. White color indicates high activation, whereas black color indicates low activation.

5.3.2 Pruning

Fig. 5.7 shows the effect of different pruning factors during training on the final classification performance. In this experiment we chose the 2-dimensional feature space, consisting of F1 and F2, because the difference between a GLIM and an HME in terms of classification performance is much more obvious. The HME consists of 8 experts, organized in a binary tree of depth 3. A pruning value of 0.0 corresponds to no pruning at all, while at a value of almost 1.0 only the most probable expert is evaluated.

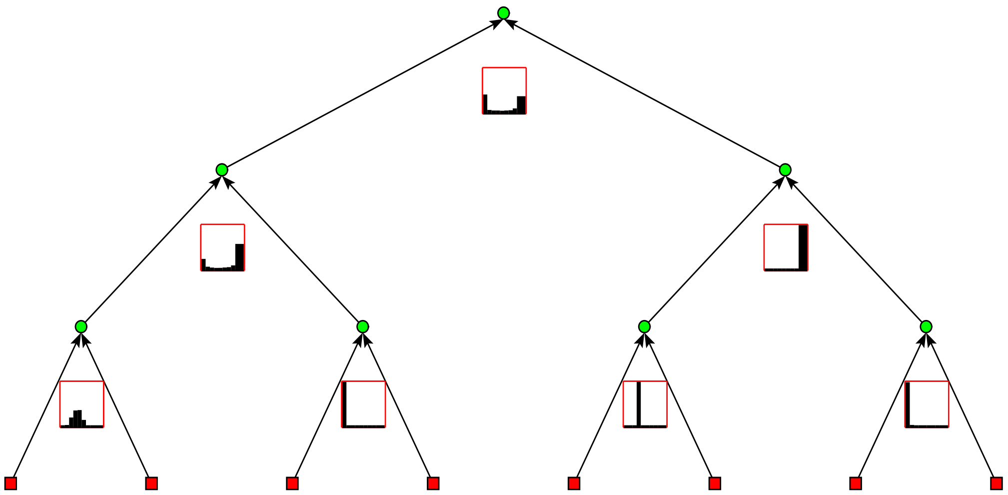

Figure 5.3: Histogram tree for a standard HME

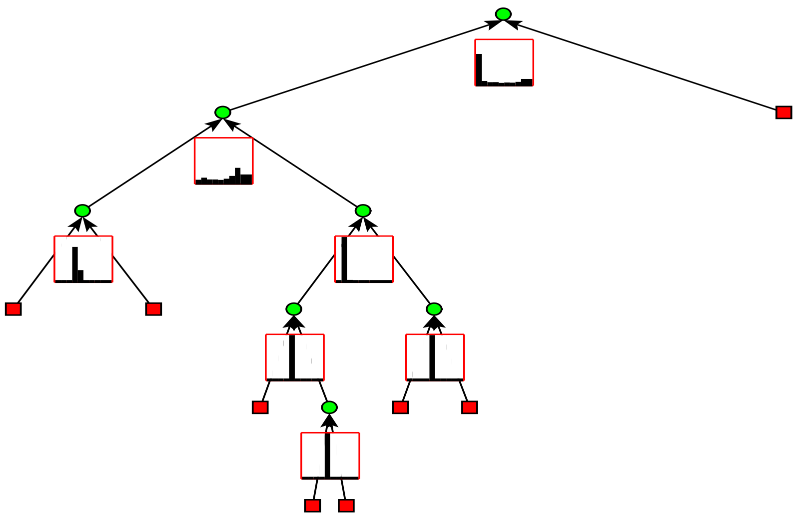

Figure 5.4: Histogram tree for a grown HME

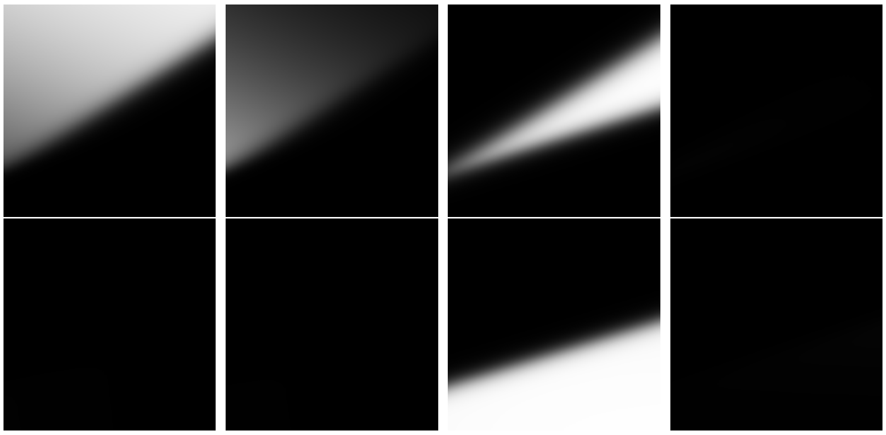

Figure 5.5: Expert activations for standard HME

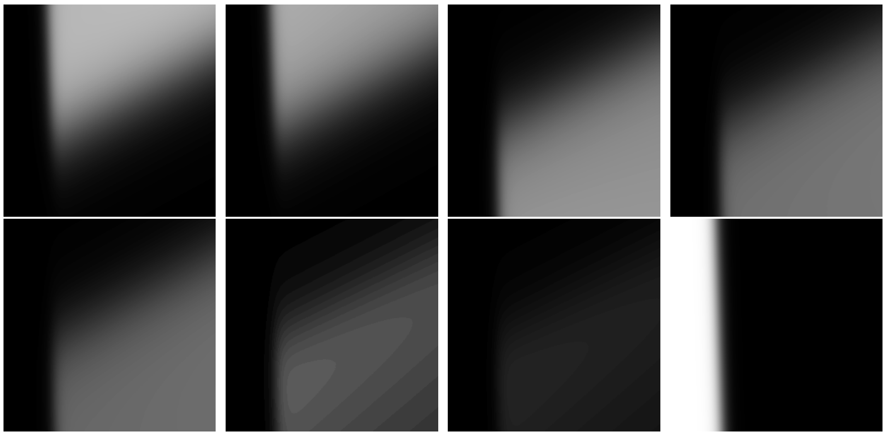

Figure 5.6: Expert activations for grown HME

Since the test set is relatively small, measuring the classification error after only one training run is not very representative, because different initial weights influence the final performance. Therefore, we computed mean and standard deviation of the classification error rate over 20 training runs, for each setting of the pruning factor. The lowest classification error rate over a maximum of 30 iterations was computed and used in each training run, although most of the training runs converged in less than 8 iterations. Finally, Fig. 5.8 shows the impact of pruning during the testing of an HME. This time, the HME was trained without pruning. Different pruning thresholds were applied during the computation of the mean square error on the test set. We chose the MSE instead of the classification rate, since the test set is too small to give significant results with respect to the classification error rate (and because GLIM and HME performances are relatively close).

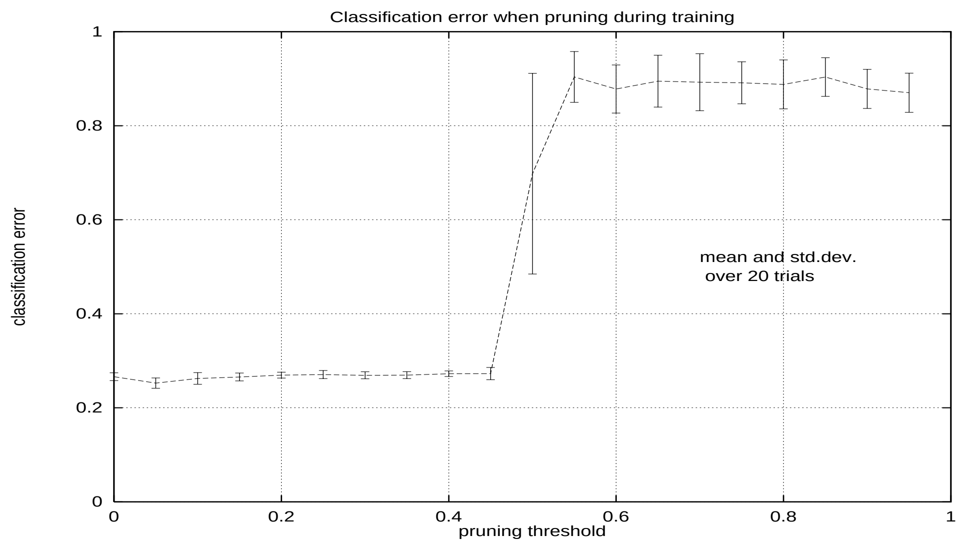

Figure 5.7: Effect of pruning during training

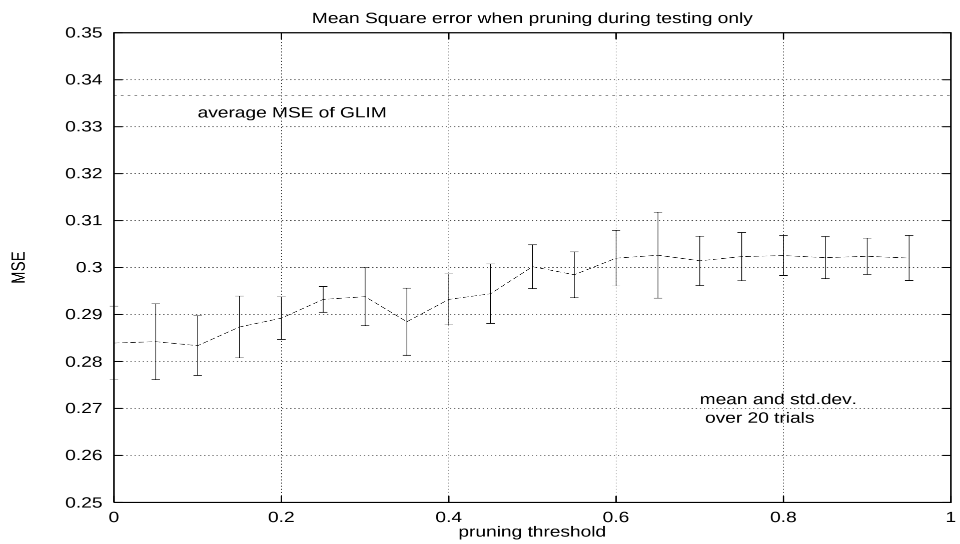

Figure 5.8: Effect of pruning during testing

Chapter 6

Context Modeling

It is well known from traditional HMM based speech recognizers, that the modeling of phonetic context improves recognition accuracy significantly over context-independent monophone models. Incorporating context models into a connectionist hybrid HMM system is also expected to boost performance, but it requires a different approach, since the computation of class likelihoods is not distributed among separate estimators, but is performed by computing class posteriors using one big neural network. This chapter introduces posterior factoring as a technique to model phonetic contexts within a hybrid connectionist speech recognizer and presents a parametric clustering algorithm that creates decision tree clustered polyphone contexts.

6.1 Phonetic Context Modeling

In a system with n monophones, modeling of context windows of width d would require the estimation of models for n^d classes, which is not feasible in practice ($n \approx 50, d > 3$). Usually, phonetic contexts are hierarchically clustered according to a distance measure between two parametric distributions. The most popular example are *generalized triphones* [32]. Systems that use this kind of modeling cluster the set of all possible/observed monophone triples (≈ 125000) into a set of about $5000 - 10000$ models. This approach, however, considers only the left and right neighbors of a monophone. More recently, systems have emerged, that cluster broader contexts, so called polyphones. Whatever the actual context modeling is, once a set of reasonable context classes is computed, it remains to estimate likelihoods for each of these classes.

A mixture of Gaussians based context-independent (CI) HMM system can be augmented to a context-dependent (CD) one fairly simple, since each class is modeled by a separate multivariate Gaussian mixture and density estimation of one context class is independent of all the other classes. As far as the acoustic modeling is concerned, it only requires a much larger set of mixture densities, the underlying mathematical framework does not restrict the number of modeled classes.

Augmenting a CI connectionist hybrid HMM system to model context classes, we are facing some difficulties, since scaled class likelihoods are computed out of class posteriors, which in turn are computed by one single neural network. This works well for a CI system with only about 50 classes, but it is computationally not feasible to model a set of over 1000 context classes by one single neural network, which would require over 1000 output neurons. Also, such a network would compute posteriors for all of the context classes in each frame, although most of them will never be used by the decoder. Training such a big network is potentially troublesome and would require too many training epochs to be applicable to speech domains with large training datasets.

6.2 Factoring Posteriors

Fortunately, posteriors for context dependent classes can be modeled by multiple neural networks, each of which containing only a small number of output neurons. Using Bayes' rule and standard rules for conditional probabilities, the context-dependent monophone likelihood $p(\mathbf{x}|c_j, \omega_i)$ for monophone ω_i and context class c_j, which is required by the HMM, can be factored in separate terms, depending on the state topology.

6.2.1 Single State Topologies

In a system where each context class is modeled by a single HMM state, the emission probability (likelihood) to be estimated in each frame is $p(\mathbf{x}|c_j, \omega_i)$. Using Bayes' rule, this is equal to

$$p(\mathbf{x}|c_j, \omega_i) = \frac{p(c_j, \omega_i|\mathbf{x})p(\mathbf{x})}{P(c_j, \omega_i)}$$

The above equation can be factored as follows using the standard rule for conditional probabilities

$$\begin{aligned} p(\mathbf{x}|c_j, \omega_i) &= \frac{p(c_j, \omega_i|\mathbf{x})p(\mathbf{x})}{P(c_j, \omega_i)} \\ &= \frac{p(c_j|\omega_i, \mathbf{x})}{P(c_j|\omega_i)} \; \frac{p(\omega_i|\mathbf{x})}{P(\omega_i)} \; p(\mathbf{x}) \end{aligned}$$

As usual, p($\mathbf{x}$) can be neglected since it is equal for all context classes c_j and all monophones ω_i given a particular frame $\mathbf{x}$, hence it will not affect the decisions made in the decoder because the $<$ relation is invariant to addition of constants.

The remaining terms in the numerators are posteriors, which can be approximated by neural nets, while the terms in the denominators are prior probabilities which can be estimated based on the frequencies of classes in the training set.

The posteriors $p(\omega_i|\mathbf{x})$ are conditioned on the input feature vector $\mathbf{x}$ only and can be approximated by a neural network which discriminates between all the monophones in the system.

The posteriors $p(c_j|\omega_i, \mathbf{x})$ are conditioned on the input feature vector and on one of the monophones ω_i. One way of estimating these probabilities, which fits neatly in the scheme of a modular neural network system, is to train separate *context expert networks* for each of the monophones. The context expert for monophone ω_i would be a network which approximates the posteriors $p_i(c_j|\mathbf{x})$ for all the context classes of monophone ω_i.

Fig. 6.1 gives an overview of a context dependent connectionist hybrid system for single state topologies.

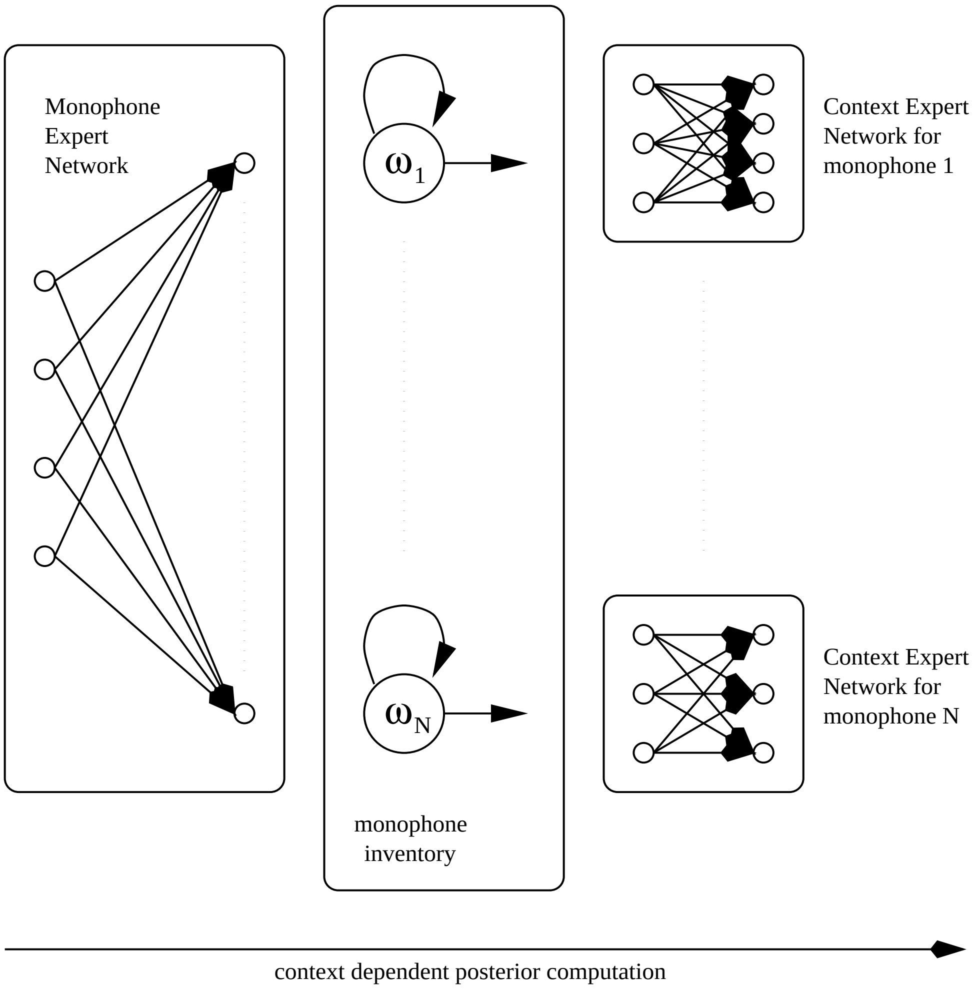

Figure 6.1: Overview: single state topology hybrid context dependent system

6.2.2 Multi State Topologies

Generally, acoustic models are made up of multiple states in a left-right or Bakis HMM model, to account for temporal variations in the modeled speech sound. Today's state-of-the-art recognizers use mostly 3-state and 5-state left-right HMMs. First, consider a context independent hybrid connectionist HMM system. There are two ways to model multi-state topologies in such a system: The first one is, to treat all the state's of all monophone models as one big pool, and train a neural network to discriminate between all of them. This approach requires $s * n$ output nodes for n monophones using s-state models. Instead, we can adhere to the concept of modularity and factor the posterior class probability further.

A multi-state HMM model requires the computation of the state, monophone and context dependent likelihood $p(\mathbf{x}|c_j, \omega_i, s_k)$, where s_k is the HMM state, c_j is the context class and ω_i is the monophone. Applying Bayes' rule and proceeding as in the case of single state models, we obtain:

$$\begin{aligned} p(\mathbf{x}|c_j, \omega_i, s_k) &= \frac{p(c_j, \omega_i, s_k|\mathbf{x})p(\mathbf{x})}{P(c_j, \omega_i, s_k)} \\ &= \frac{p(c_j, \omega_i|s_k, \mathbf{x})}{P(c_j, \omega_i|s_k)} \; \frac{p(s_k|\mathbf{x})}{P(s_k)} \; p(\mathbf{x}) \\ &= \frac{p(c_j|\omega_i, s_k, \mathbf{x})}{P(c_j|\omega_i, s_k)} \; \frac{p(\omega_i|s_k, \mathbf{x})}{P(\omega_i|s_k)} \; \frac{p(s_k|\mathbf{x})}{P(s_k)} \; p(\mathbf{x}) \end{aligned}$$

All the terms in the denominators are again prior probabilities, which we can estimate by relative frequencies. The frame probability $p(\mathbf{x})$ can be dropped, when seeking the model with maximum likelihood. It remains to compute the posteriors in the numerators.

Starting from the right side, the posteriors $p(s_k|\mathbf{x})$ can be computed by a single neural network, discriminating between the states in a s-state HMM topology. Therefore, we call this network a *state discriminating network (SDN).*

The posteriors $p(\omega_i|s_k, \mathbf{x})$ are conditioned on the HMM state and the input frame and can be computed by a set of s networks, one for each HMM state. These networks discriminate between the monophones ω_i, given a particular HMM state s_k. The network for state s_k computes $p_k(\omega_i|\mathbf{x})$.

The posteriors $p(c_j|\omega_i, s_k, \mathbf{x})$ are conditioned on the input frame $\mathbf{x}$, the HMM state s_k and the monophone ω_i. They can be computed by a matrix of networks consisting of s times n networks (s is the number of states, n is the number of monophones). Each of these networks discriminates between all the context classes of a specific monophone in a specific state. The network for state s_k and monophone ω_i therefore computes $p_{ki}(c_j|\mathbf{x})$.

Fig. 6.2 gives an overview of a context dependent connectionist hybrid system for multi state topologies.

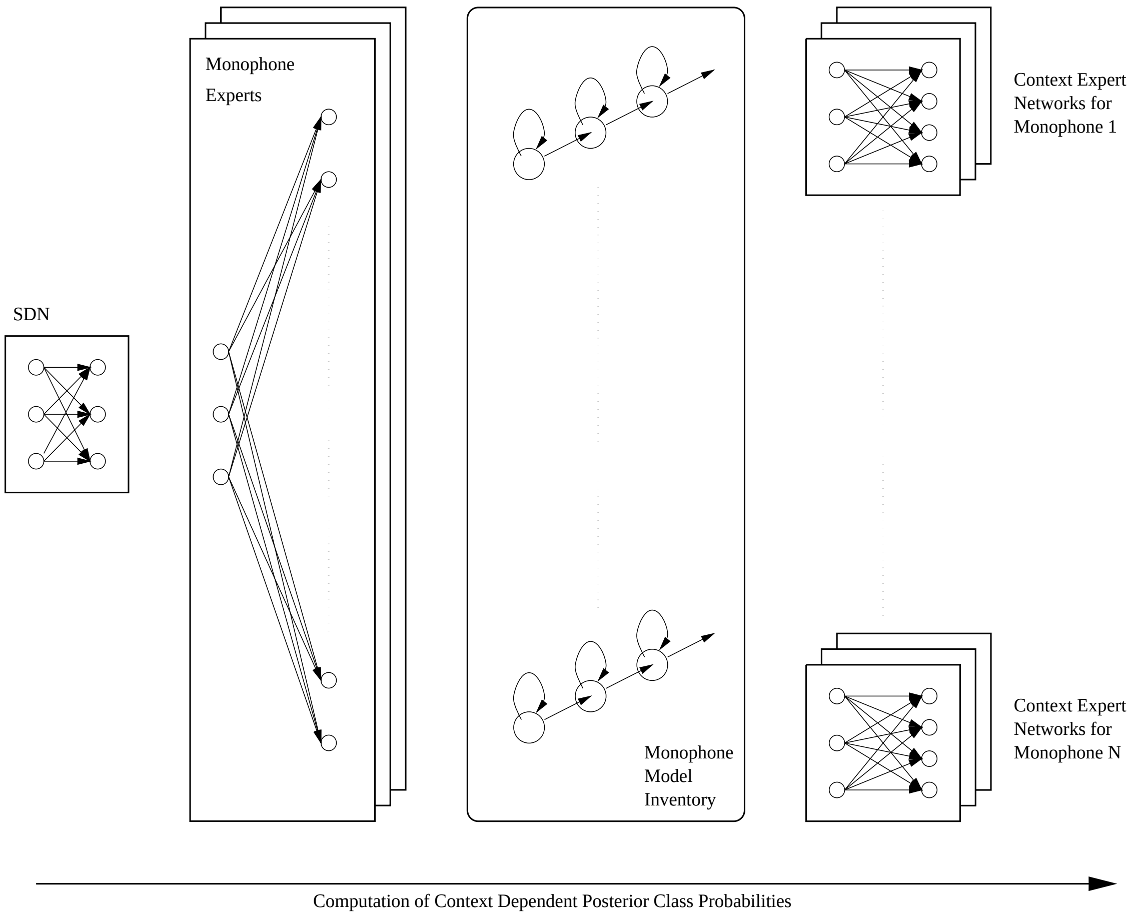

Figure 6.2: Overview: multi state topology hybrid context dependent system

The networks depicted in Fig. 6.1 and Fig. 6.2 look like single layer perceptrons, but they are meant to represent arbitrary posterior probability estimators. Computation of a specific context dependent likelihood $p(\mathbf{x}|c_j, \omega_i, s_k)$ requires the evaluation of three networks: The state discriminating network (SDN), one of the monophone expert networks and one of the context expert networks. Note, that the context-dependent hybrid connectionist system can easily be switched back to context-independent (CI) mode by turning off the context expert networks, a feature not available in mixture-of-Gaussians based systems.

6.2.3 Related Work

The modeling of context dependent likelihoods as presented in this thesis most closely resembles the work in [30] and [31], with the noteable difference, that we have generalized context-dependent posteriors to multi-state HMM models.

There are other ways of factoring a conditional posterior probability. For instance, one could decompose the conditional likelihood for a one-state HMM model as follows:

$$
\begin{aligned}
p(\mathbf{x}|c_j, \omega_i) &= \frac{p(c_j, \omega_i|\mathbf{x})p(\mathbf{x})}{P(c_j, \omega_i)} \\
&= \frac{p(\omega_i|c_j, \mathbf{x})}{P(\omega_i|c_j)} \quad \frac{p(c_j|\mathbf{x})}{P(c_j)} \quad p(\mathbf{x})
\end{aligned}
$$

In this case, context specific networks are trained to discriminate between the monophones ω_i, given a specific context class c_j. Every context specific network performs a simpler task than a context-independent network. This approach is adopted by SRI [13]. However, it is less attractive to us, because of the following two reasons: (1) One can not switch between CI and CD mode and (2) discriminating between monophones in a specific context can lead to poor posterior estimates, when some monophones occur rarely or not at all in this context. Furthermore, as we will see in the next chapter, our approach of factoring posteriors allows to make use of the same context clustering trees that are used in mixture-of-Gaussian based HMM systems.

Yet another approach was adopted by Bourlard and Morgan at ICSI [3]. Their method factors the posterior phone-in-context probability in the same way as we presented it. However, their system uses only one MLP to estimate context posteriors instead of a set of context experts as proposed earlier in this thesis. This is possible by giving the context MLP extra binary inputs, which encode the current monophone. This approach has the disadvantage of requiring multiple forward passes through the context MLP during recognition, since the decoder will hypothesize more than one monophone at each time step, which leads to different network input patterns.

6.3 Polyphone Clustered Contexts

We have presented an architecture for estimating context dependent posterior monophone probabilities, given a set of context classes. We have not yet talked about how we obtain these contextual classes. The remainder of this chapter will present polyphone clustering using decision trees, as it is used within the mixture-of-Gaussians based JANUS recognizer. We will show, that the resulting context clustering trees can also be used to derive phonetic context classes for the context expert networks in our hybrid framework.

6.3.1 Polyphones

Polyphones are generalizations of the well-established triphones. They model a broader context of a given monophone. For instance, the word 'BABYSITTING' is modeled, according to our dictionary, as the following sequence of monophones:

B - EY - B - IY - S - IH - DX - IX - NG

If we'd model, for instance, polyphonic contexts of a maximum of $+/-2$ phones, the above word would be modeled as a sequence of the following polyphones:

Monophone	Polyphone
B	* - * - B - EY - B
EY	* - B - EY - B - IY
B	B - EY - B - IY - S
IY	EY - B - IY - S - IH
S	B - IY - S - IH - DX
IH	IY - S - IH - DX - IX
DX	S - IH - DX - IX - NG
IX	IH - DX - IX - NG - *
NG	DX - IX - NG - * - *

An inventory of polyphones can be extracted from large text corpora and stored efficiently in a set of binary decision trees, one for each monophone. It should be obvious, that the number of polyphones observed in a given large text corpus is far too high to allow separate models for each one of them. In fact, many of the observed polyphones do occur only once in the training set. Additionally, there may be some polyphones in an unseen test corpus, which were not present in the training corpus, no matter how big the latter was.

Therefore, we need to apply a clustering procedure, which reduces the number of distinct models while providing full coverage of unseen new test data. By far the most popular technique is to use decision trees with questions about the phonetic context. Decision trees are very appealing because they guarantee to cover all phones in any contexts, while using a distance measure based on the acoustic data to split nodes and grow the tree.

6.3.2 Decision Tree Clustering

Decision trees are divisive clustering methods making use of binary trees asking questions at each internal node. Associated with a decision tree is a finite set of questions which can be answered with yes or no. The children nodes of each internal node correspond to the two possible answers to the particular question asked. Starting with a tree containing only the root node, succesive splits are applied to grow the tree to a desired size.

The iterative tree growing procedure works as follows: Initially, all the acoustic training data is associated with the root node. In each growing step, a preliminary split is computed for all of the leave nodes and all the possible questions, that can be asked. Each of these preliminary splits is scored using a distance measure which models the goodness of the split. The leave node with the best score is then split, while all the other preliminary splits are discarded. The training data associated with the node being split, is distributed among the children nodes according to the answers to the actual question being used. The distance measure used to score the preliminary node splits is very much dependent on the representation of the data. In [30],[31], unimodal multivariate Gaussians with diagonal covariance matrices are used to model the data in each leave node. They use the gain in log-likelihood due to the data being split as the distance measure. This involves the estimation of diagonal covariance matrices for each hypothesized node split:

$$\Delta L = n \log |\Sigma| - (n_l \log |\Sigma_l| + n_r \log |\Sigma_r|)$$

where n is the number of samples associated with the parent node, n_l and n_r are the number of samples associated with the children nodes, respectively, Σ is the diagonal covariance matrix of the data in the parent node and Σ_l and Σ_r are the diagonal covariance matrices of the data in the children nodes, respectively.

Once a decision tree for a particular monophone is grown to a desired size, its leaves represent the context classes of that monophone and are labeled accordingly.

6.3.3 Entropy based Clustering

The distance measure used in [30],[31] requires the estimation of covariance matrices for each hypothesized node split using all the acoustic data associated with the nodes involved in the split. This can be very expensive, especially when the training dataset and the set of questions are large.

Phonetic context decision trees in JANUS are grown using a distance measure that does not depend on the acoustic training data directly. Instead, the mixture coefficients of the context independent Gaussian mixtures are interpreted as discrete distributions over a vector quantized feature space, represented by the codebooks of Gaussians. When hypothesizing a new split, discrete distributions over the same monophone codebook are computed for the two hypothesized children nodes. To score the goodness of the split, the gain in entropy using separate distributions for the children nodes is computed.

$$\begin{aligned}
D(\mathbf{p}, \mathbf{p}_l, \mathbf{p}_r) &= n_l H_l(\mathbf{p}_l) + n_r H_r(\mathbf{p}_r) - n H(\mathbf{p}) \\
\text{with} \qquad H_l(\mathbf{p}_l) &= -\sum_i p_{li} \log p_{li} \\
H_r(\mathbf{p}_r) &= -\sum_i p_{ri} \log p_{ri} \\
H(\mathbf{p}) &= -\sum_i p_i \log p_i
\end{aligned}$$

In the above equation, the sums go over the number of coefficients in the discrete probability distributions. Using the above distance function to score splits in a decision tree is efficient and appealing from an information theoretic point of view, since the above splitting score can be interpreted as the mutual information between children nodes distribution.

6.3.4 Analyzing Cluster Trees

To show properties of the splitting criterion, we created cluster trees for the ESST speech task with 5 different numbers of overall context models : 500, 1000, 1500, 2000 and 2500. The ESST speech task is an English spontaneous speech database which we also use for the evaluation of the hybrid speech recognizer (see Chapter 8 for details).

For each of the 5 cluster trees, we computed the number of context models generated for each monophone (over all states of a 3-state left-right HMM model). Fig. 6.3 shows the evolution of the number of context models over the 5 cluster phases and the 52 monophones in our system.

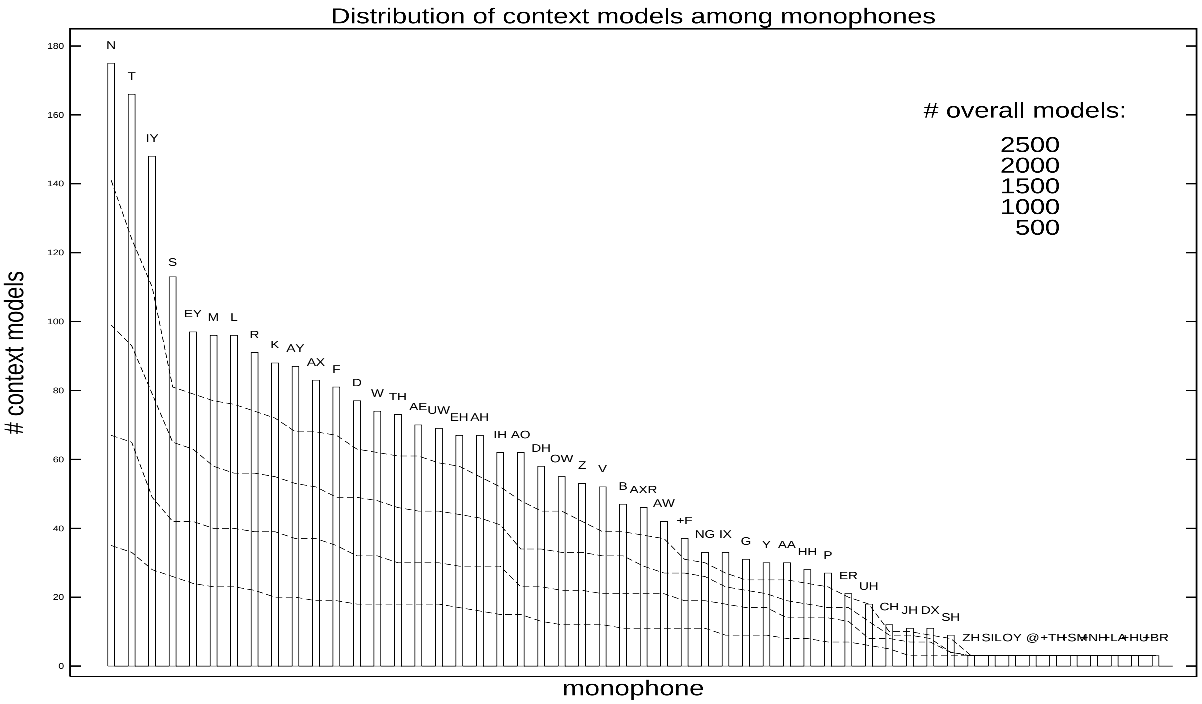

Figure 6.3: Distribution of context models

It is remarkable that the trees for the monophones N, T and IY together contain about 20% of all cluster models (2500) over all trees. Fig. 6.4 shows a typical decision tree. It was build for the middle state of a three-state model of the monophone AX. It is part of a forest of 156 decision trees (52 monophones times 3 states) with an overall number of

1000 context models. The polyphonic context is restricted to the 3 phones left and right of a midphone. The set of all possible questions, that were available for the generation of the tree is listed in Appendix A.

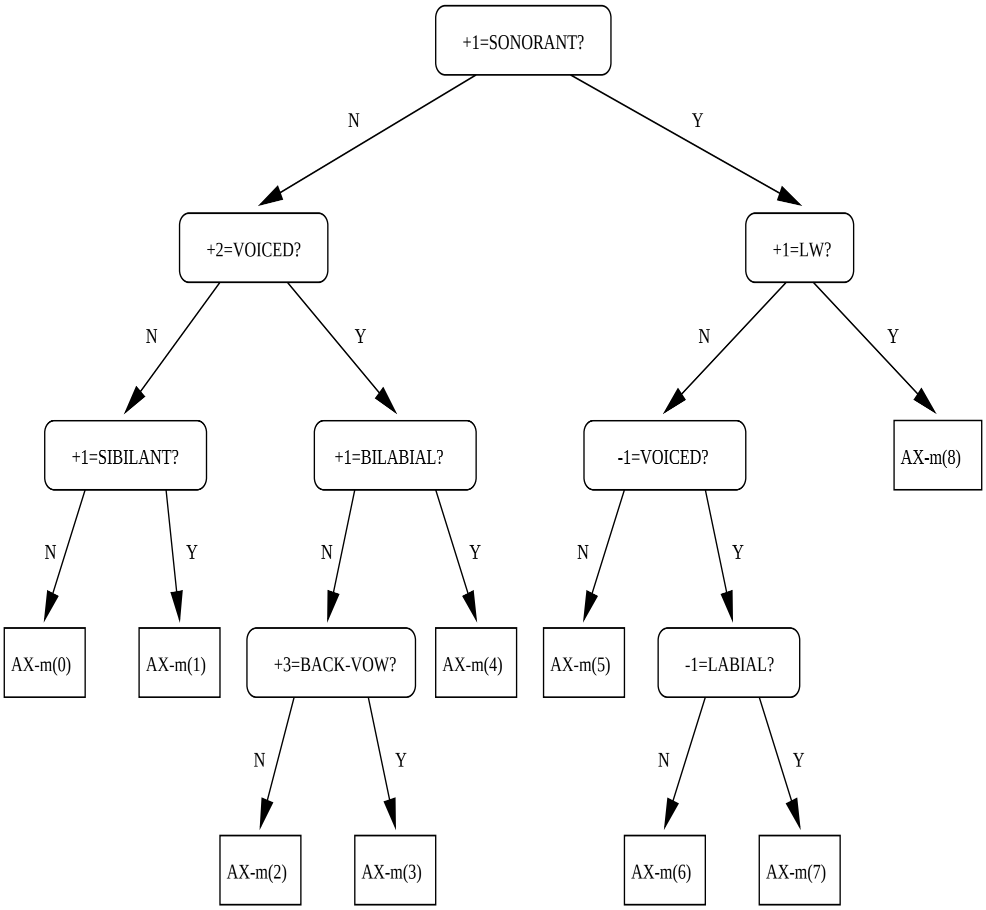

Figure 6.4: Decision tree for monophone AX-m

Obviously, the clustering process favours questions about the immediate right or left neighboring phone. This is consistent with our intuition that the influence of context is decreasing with increasing neighborhood distance. Nevertheless, the tree in Fig. 6.4 also uses questions about the broader context. It even asks a question about a phone that lies 3 frames in the future, although such questions generally occur only in the lower parts of the trees. That means, that it is in fact helpful to consider broader contexts than just triphones. In the beginning of node splitting, the tree concentrates on neighboring contexts, but when the trees get bigger, the splitting process starts to use broader context questions as well.

Chapter 7

Mixtures of Gaussian Experts

Until now, we have assumed a generalized linear model in both gates and experts of an Hierarchical Mixture of Experts, although the architecture in principle allows arbitrary parametric forms of gates and experts. In the case of classification, however, the models for gates and experts have to fullfil the constraint, that their output activations sum up to one for each input frame. Recently, Xu, Jordan and Hinton [57] have proposed to use a parametric form based on Gaussian kernels for the gates. We will further develop their work, showing that the same parametric form can be used for experts as well. Such an architecture is very attractive because it can be initialized to a near optimal solution very efficiently, thus reducing convergence time of the learning algorithm.

7.1 Alternative Parameterization

Instead of applying a generalized linear model with softmax nonlinearity, the following parameterization was proposed for the gate in a one-level mixture of experts architecture ([57]):

$$
\begin{aligned}
g_i(\mathbf{x}, \mathbf{v}) &= \frac{\alpha_i P(\mathbf{x}|\mathbf{v_i})}{\sum_k \alpha_k P(\mathbf{x}|\mathbf{v_k})} \qquad \text{with} \qquad \sum_k \alpha_k = 1 \quad \text{and} \quad \alpha_k \geq 0 \\
P(\mathbf{x}|\mathbf{v_i}) &= \frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \exp\left\{-1/2(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i)\right\}
\end{aligned}
$$

This form of a gate is legal, since the g_i's by definition sum up to one, thus providing a partition of unity for each input feature vector $\mathbf{x}$. The above parametric form can be interpreted as a parametric a-posteriori classifier according to Bayes theorem:

$$p(\omega_i|\mathbf{x}) = \frac{P(\omega_i)p(\mathbf{x}|\omega_i)}{\sum_k P(\omega_k)p(\mathbf{x}|\omega_k)}$$

where the prior probabilities $P(\omega_i)$ are the α_i's and the class likelihoods $p(\mathbf{x}|\omega_i)$ are modeled by single Gaussian distributions.

7.2 Gaussian Classifier as Gate

Parameterizing the gate of a mixture of experts as a Gaussian a-posteriori classifier allows to derive an efficient single-loop EM algorithm to estimate the parameters of the gate. Additionally, the special parametric form allows to initialize the Gaussian kernels and a-priori probabilities which speeds up training times significantly.

7.2.1 EM algorithm

The conditional mixture underlying a mixture of experts is

$$\begin{aligned} P(\mathbf{y}|\mathbf{x},\mathbf{\Theta}) &= \sum_i g_i P_i(\mathbf{y}|\mathbf{x},\mathbf{\Theta_i}) \\ &= \sum_i \frac{\alpha_i P(\mathbf{x}|\mathbf{v_i})}{\sum_k \alpha_k P(\mathbf{x}|\mathbf{v_k})} P_i(\mathbf{y}|\mathbf{x},\mathbf{\Theta_i}) \\ &= \sum_i \frac{\alpha_i P(\mathbf{x}|\mathbf{v_i})}{P(\mathbf{x},\mathbf{v})} P_i(\mathbf{y}|\mathbf{x},\mathbf{\Theta_i}) \end{aligned}$$

If we attempt to derive an EM algorithm directly on this mixture density, we find that the M-step is not analytically solvable and would require iterative processing, similar to the IRLS algorithm. However, the above conditional mixture can be rewritten in a form, that allows an analytical solution for the ML problem:

$$P(\mathbf{y},\mathbf{x}) = P(\mathbf{y}|\mathbf{x},\mathbf{\Theta})P(\mathbf{x},\mathbf{v}) = \sum_i \alpha_i P(\mathbf{x}|\mathbf{v_i}) P_i(\mathbf{y}|\mathbf{x},\mathbf{\Theta_i})$$

Instead of estimating the gating parameters to maximize the likelihood of the original mixture density, we can maximize the likelihood of the above joint density. Applying the EM algorithm in a similar way as we did in the case of generalized linear models leads to the following iterative estimation method:

(1) E-step For each training vector, compute the posterior node probabilities h_i according to

$$h_i^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}) = \frac{\alpha_i^{(j)} P(\mathbf{x}^{(t)}|v_i^{(j)}) P_i(\mathbf{y}^{(t)}|\mathbf{x}^{(t)},\Theta_i^{(j)})}{\sum_k \alpha_k P(\mathbf{x}^{(t)}|v_k^{(j)}) P_k(\mathbf{y}^{(t)}|\mathbf{x}^{(t)},\Theta_k^{(j)})}$$

(2) M-step Use the h_i's to compute new estimates for the parameters α_i, μ_i and Σ_i of the gate. The new estimates can be computed directly, since the ML problem is now analytically solvable:

$$\begin{aligned}
\alpha_i^{(j+1)} &= \frac{\sum_t h_i^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})}{\sum_t \sum_k h_k^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})} \\
\mu_i^{(j+1)} &= \frac{\sum_t h_i^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})\mathbf{x}^{(t)}}{\sum_t h_i^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})} \\
\Sigma_i^{(j+1)} &= \frac{\sum_t h_i^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}) \left[\mathbf{x}^{(t)} - \mu_i^{(j+1)}\right] \left[\mathbf{x}^{(t)} - \mu_i^{(j+1)}\right]^T}{\sum_t h_i^{(j)}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})}
\end{aligned}$$

The ML problem for the experts remains analytically unsolvable (in the case of classification) and those parameters must be estimated either iteratively by gradient ascent or by the least squares heuristic. However, the above EM algorithm for gates is computationally more efficient than the IRLS algorithm for GLIMs. Note, that the computation of node posteriors h_i has changed compared to the EM algorithm for GLIMs. This indirectly influences the estimation of expert parameters also, since the joint node posteriors appear in the re-estimation formulas for experts.

Note also, that the above formulation of the EM learning does maximize the sum of the mixture likelihood and the conditional likelihood of the gate instead of maximizing the mixture likelihood itself. During testing, however, the output of the mixture still follows the mixture model of HME's.

7.2.2 Initialization

The parametric form which we have applied to the gate is very attractive because it allows the initialization of parameters to near optimal values. There is a significant body of work on the initialization of Gaussian mixture models and radial basis function networks which can be adopted here as well. In fact, since we already know, that the parametric form can be viewed as a Gaussian a-posteriori classifier, its parameters can best be initialized by estimating priors and class likelihoods by relative frequencies and maximum likelihood estimation, respectively. However, in the case of a gate in a mixture of experts, we do not have class labels to estimate the parameters of a Gaussian classifier the way we just proposed (nevertheless, this technique will gain importance later, when we'll use Gaussian classifiers as experts also).

One possible initialization technique for Gaussian gates that works very well in practice is to estimate the parameters such that the likelihood of the data under an unsupervised mixture model is maximized. That means, we initalize the parameters of the gate according to

$$\hat{v}_i = \arg\max_{v_i} \sum_t \log \sum_i \alpha_i^{(t)} P(\mathbf{x}^{(t)}|\mathbf{v_i})$$

with $\sum_k \alpha_k = 0$ and $\alpha_k \geq 0$. Usually, maximizing such a likelihood is done in the following three steps:

(1) Extract Samples Initialize the means of the Gaussians by extracting the appropriate number of samples randomly from the training set.

(2) Cluster Means Apply a clustering algorithm such as the k-means or LBG algorithm to the means. This corresponds to minimizing the distortion of a discrete vector-quantized distribution where the codebook vectors are the means.

(3) Maximum Likelihood Iteratively reestimate the mixture coefficients α_i, the means μ_i and the covariance matrices Σ_i according to the EM algorithm for Gaussian mixtures [10].

The possibility to initalize the gate parameters to near optimal solutions and the single-loop EM re-estimation algorithm render the Gaussian parameterization a powerful extension to the standard HME architecture.

7.2.3 Combining Multiple Classifiers

There is one other application of Gaussian gates, namely the task of combining multiple classifiers (CMC). Suppose we have n different kind of pre-trained classifiers, all trained on the same data set. Since each of the classifiers might have learned different parts of the data best, it is generally a good idea to combine their estimates, if we have a combination method capable of supporting the good and suppressing the bad classifiers for each training sample.

The problem can be treated as a special case of a mixture of experts, where the experts parameters remain fixed and only the gates are iteratively adapted. The single-loop EM algorithm can therefore be directly used to estimate the gate parameters. It was shown in [57] that this can increase overall performance considerably, while avoiding the costly re-estimation of the expert classifiers. This makes this technique even more attractive for our purpose in speech recognition, since we have to deal with large datasets consisting of millions of feature vectors.

7.3 Mixture of Gaussian Experts

Given the advantages of the Gaussian parameterization of the gate, it would be nice, if we could use the same parameterization for the experts as well. Also, we would like to generalize the technique to *hierarchical* mixtures with more than one gate. Unfortunately, the solution to the EM learning problem proposed in [57] does not generalize to experts. We will therefore relax the EM constraint and derive a *generalized* EM algorithm that only guarantees to increase the mixture likelihood in each iteration, instead of maximizing it.

7.3.1 Gaussian Classifiers as Experts

The parametric form based on Gaussian kernels is even more attractive for experts than it is for gates. The reason is, that in the case of experts, we have class labels for the initialization available. This simplifies the initialization of expert parameters, since each Gaussian kernel can be estimated independently on a subset of the data. Given that the gate is already initialized, the initalization of the experts requires just a single pass through the training data, yet yielding parameter estimates which give the mixture an initial performance that is close to the optimal one, even before applying any kind of training algorithm to the whole architecture.

7.3.2 GEM algorithm

As promised, we will now derive a generalized EM algorithm for a mixture of experts which uses Gaussian parameterizations exclusively. The probability model of the overall architecture is

$$P(\mathbf{y}|\mathbf{x},\mathbf{\Theta}) = \sum_i g_i(\mathbf{y}|\mathbf{x},\mathbf{v_i}) P_i(\mathbf{y}|\mathbf{x},\mathbf{\Theta_i})$$

where the P_i are multinomial densities, modeling the multiway classification task imposed on the experts and the $\mathbf{v_i}$ and $\mathbf{\Theta_i}$ are the sets of parameters for gate and experts, respectively. The expert activations are computed the same way as the gate activations, assuming a Gaussian a-posteriori classifier:

$$\begin{aligned} y_{ij}(\mathbf{x},\mathbf{\Theta}_i) &= \frac{\alpha_{ij} P(\mathbf{x}|\mathbf{\Theta}_{ij})}{\sum_k \alpha_{ik} P(\mathbf{x}|\mathbf{\Theta}_{ik})} \qquad \text{with} \qquad \sum_k \alpha_{ik} = 1 \quad \text{and} \quad \alpha_{ik} \geq 0 \\ P(\mathbf{x}|\mathbf{\Theta_{ij}}) &= \frac{1}{(2\pi)^{n/2}|\Sigma_{ij}|^{1/2}} \exp\left\{-1/2(\mathbf{x}-\mu_{ij})^T \Sigma_{ij}^{-1} (\mathbf{x}-\mu_{ij})\right\} \end{aligned}$$

The expert activations can be re-written in an interesting form:

$$y_{ij}(\mathbf{x},\mathbf{\Theta}_i) = \frac{\exp(z_{ij})}{\sum_k \exp(z_{ik})}$$

$$\text{with} \qquad z_{ij} = \log(\alpha_{ij}) - \frac{1}{2}\left[n\log(2\pi) + log|\Sigma_{ij}| + (\mathbf{x}-\mu_{ij})^T \Sigma_{ij}^{-1} (\mathbf{x}-\mu_{ij})\right]$$

We have expressed the expert activations using the same 'softmax' nonlinearity as in the GLIM case. The difference is, that we changed the underlying linear model which computes $z = WX$ to a radial model, which basically computes $z = (X - W)^2$. Expressing the new model in terms of the softmax function allows us to unify linear and radial expert models.

The M-step of the EM algorithm for mixtures of experts involves the maximization of the following two likelihoods (assuming a multinomial probability model)

$$\begin{aligned}
\mathbf{v}_i^{(k+1)} &= \arg\max_{\mathbf{v}_i} \sum_t \sum_j h_j^{(t)} \log g_j^{(t)} \\
\theta_i^{(k+1)} &= \arg\max_{\theta_i} \sum_t h_i^{(t)} \sum_j t_j^{(t)} \log y_{ij}^{(t)}
\end{aligned}$$

where the $t_j^{(t)}$ are targets for the expert output nodes. Because of the nonlinearity of the softmax function in both g and μ, there is no closed-form solution to this problem. Therefore, we derive a GEM algorithm which increases the likelihoods using gradient ascent

$$\begin{aligned}
\mathbf{v}_i^{(k+1)} &= \mathbf{v}_i^{(k)} + \eta \sum_t \left[\sum_j h_j^{(t)} (\delta_{ij} - g_j^{(t)}) \right] \frac{\partial z_i}{\partial v_i} \\
\theta_{ij}^{(k+1)} &= \theta_{ij}^{(k)} + \eta \sum_t h_i^{(t)} \left[\sum_l t_l^{(t)} (\delta_{jl} - y_{il}^{(t)}) \right] \frac{\partial z_j}{\partial \theta_j}
\end{aligned}$$

where δ_{ij} is the Kronecker symbol, η is the learning rate, and the z_i are the linear or radial functions prior to the softmax nonlinearity.

In the case of Gaussian experts with diagonal covariance matrices, we obtain the following update rules for the parameters of a specific expert E_i:

$$\begin{aligned}
\alpha_j^{(k+1)} &= \alpha_j^{(k)} + \eta \sum_t h_i^{(t)} \left[\sum_l t_l^{(t)} (\delta_{jl} - y_l^{(t)}) \right] \frac{1}{\alpha_j^{(t)}} \\
\mu_{jm}^{(k+1)} &= \mu_{jm}^{(k)} + \eta \sum_t h_i^{(t)} \left[\sum_l t_l^{(t)} (\delta_{jl} - y_l^{(t)}) \right] \frac{(x_m - \mu_{jm}^{(t)})}{\sigma_{jm}^{2(t)}} \\
\sigma_{jm}^{2(k+1)} &= \sigma_{jm}^{2(k)} + \eta \sum_t h_i^{(t)} \left[\sum_l t_l^{(t)} (\delta_{jl} - y_l^{(t)}) \right] \frac{(x_m - \mu_{jm}^{(t)})^2 - \sigma_{jm}^{2(k)}}{\sigma_{jm}^{4(t)}}
\end{aligned}$$

The α_j's need to be normalized after each iteration, in order to fulfill the constraint, that their sum yields one. To speed up convergence, it is possible to use this algorithm in a stoachastic gradient based version, updating the parameters each time M training samples have been presented. The presented GEM algorithm is basically a first-order technique, therefore, the reader may argue that convergence speed might be too slow to render this algorithm useful. However, we will show, that the combination of this algorithm with the initalization technique presented above yield very fast convergence in practice.

7.3.3 BBI Trees for Pruning

In [16], we presented a binary tree based space partioning algorithm which is very effective in speeding up the evaluation of Gaussian mixtures with diagonal covariance matrices. This algorithm partitions the feature space in a set of 2^d so called *buckets* by means of hyperplanes orthogonal to one of the coordinate axis. Given a particular feature vector $\mathbf{x}$, the algorithm is able to determine the bucket, in which the vector resides, with just a few scalar comparisons. Having determined the correct bucket, a reduced list of Gaussians, which is computed in advance, is evaluated instead of the whole mixture.

This algorithm can easily be applied to speed up a Gaussian classifier based hierarchical mixture of experts, if the diagonal covariance assumption holds. First, we compute a BBI space partioning tree for each of the Gaussian classifiers (each node in the MGE tree). During training or testing, when the MGE nodes are asked to compute posterior probabilities, the BBI trees are used to determine a reduced set of Gaussians, which contribute more than a specific threshold. Only these Gaussians are then evaluated, all the remaining ones are pruned to an activation of 0.0. This technique can be seen as a form of MGE tree pruning, if applied to gating nodes, where each Gaussian in the gate classifier corresponds to one of the children nodes. We found that BBI trees for MGE pruning are particularly useful for MGE topologies with a high branching factor. The overhead of pre- computing BBI trees for each MGE node is neglectable during the training of MGE's. For testing, the BBI trees only have to be computed once and can be stored together with the remaining MGE tree parameters.

7.4 Experiments

We trained a GLIM- and a Gauss-classifier based mixture of experts on the Peterson & Barney vowel data, to compare the two parameterizations. The architecture was the same in both cases, a 1-level tree, featuring 1 gate and 10 experts. We chose the branching factor of the tree to be the number of output classes, because this allows an even faster initialization scheme for the MGE than presented so far. Initialization for the MGE proceeds in two steps (requiring two iterations through the training data):

(1) Estimate parameters of a single Gaussian expert. Expand the tree to a 1-level, 10 children architecture, switching the Gaussian expert to a Gaussian gate and freeze its parameters.

(2) Estimate parameters of the 10 new experts, using the gate activations as observation weights.

After the initalization, we train the architecture using the GEM algorithm, presented earlier. Fig. 7.1 shows the log-likelihood on the training set for an MGE and an HME. Fig. 7.2 shows the mean square error on the test set for the same training run.

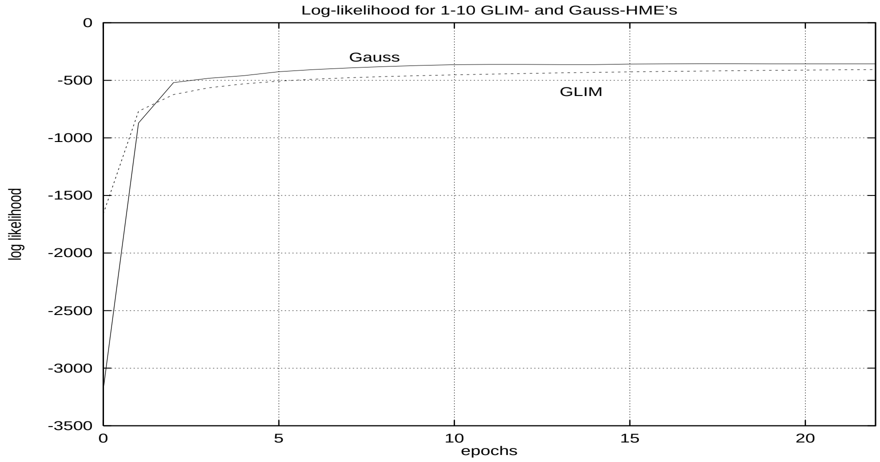

Figure 7.1: Evolution of log-likelihood for HME and MGE during training

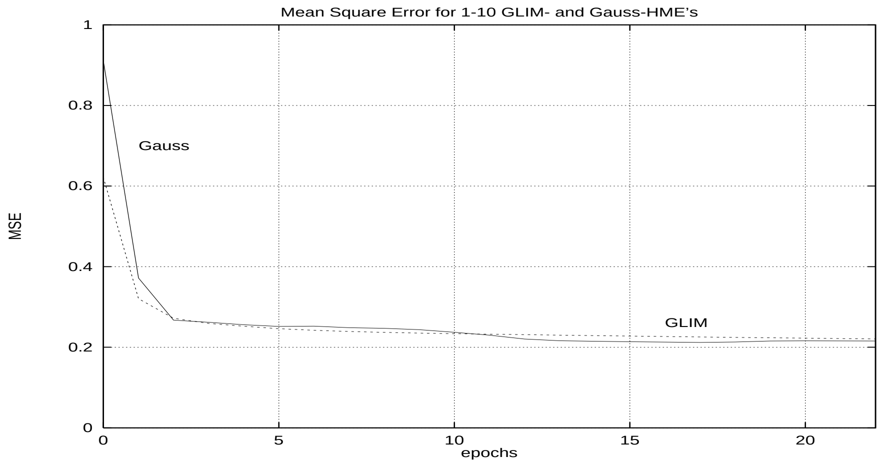

Figure 7.2: Evolution of MSE for HME and MGE during trainig

The first two iterations for the MGE consist of initializing the parameters. The performance of the MGE after initialization is already very high, yet the following GEM training can improve performance further. Note, that the initialization phase for the MGE is taking considerably less time than a regular GEM or EM iteration, where we have to compute node and branching posteriors. Taking this into account, the MGE compares favourably to a same-size HME.

Chapter 8

Evaluation

8.1 Hybrid Janus

This section briefly introduces the hybrid HME/HMM speech recognition system, that was developed during this thesis. As a starting point of this work, there was a fully functional continuous-density HMM speech recognizer available - JANUS-SR version 3. This system integrates the basic recognizer modules, such as feature extraction, acoustic modeling, language modeling and the decoder. The goal of this thesis was, to implement a complete new acoustic scoring module based on HME's for JANUS, which can be used stand-alone or in combination with the existing mixture-of-Gaussians scoring module. Version 3 of the JANUS recognizer was constructed as a speech recognition toolbox, exporting all the relevant data structures and methods in an object oriented fashion, using the Tcl/Tk toolkit as the user front-end.

8.1.1 General Concept

The JANUS recognizer implements acoustic scoring by a generic object, called 'stream'. A system can contain one or more of such streams. Each stream can be trained and asked for estimates of model likelihoods. One important concept in JANUS is, that the streams are responsible for the modeling of basic acoustic units. All other modules interface with the streams by tagged sequences of phones. This allows the use of different context-models by different streams and facilitates the integration of a connectionist score computer. For instance, a tied-state continuous density mixture-of-Gaussians scoring with typically about 5000 context models can easily be combined with a context-independent connectionist a-posteriori scoring.

The hybrid system, developed for this thesis, allows context-independent and context-dependent connectionist (HME) scoring of multi-state HMM's, using decision trees to cluster models. Fig. 8.1 gives an overview of the connectionist part of the hybrid JANUS system.

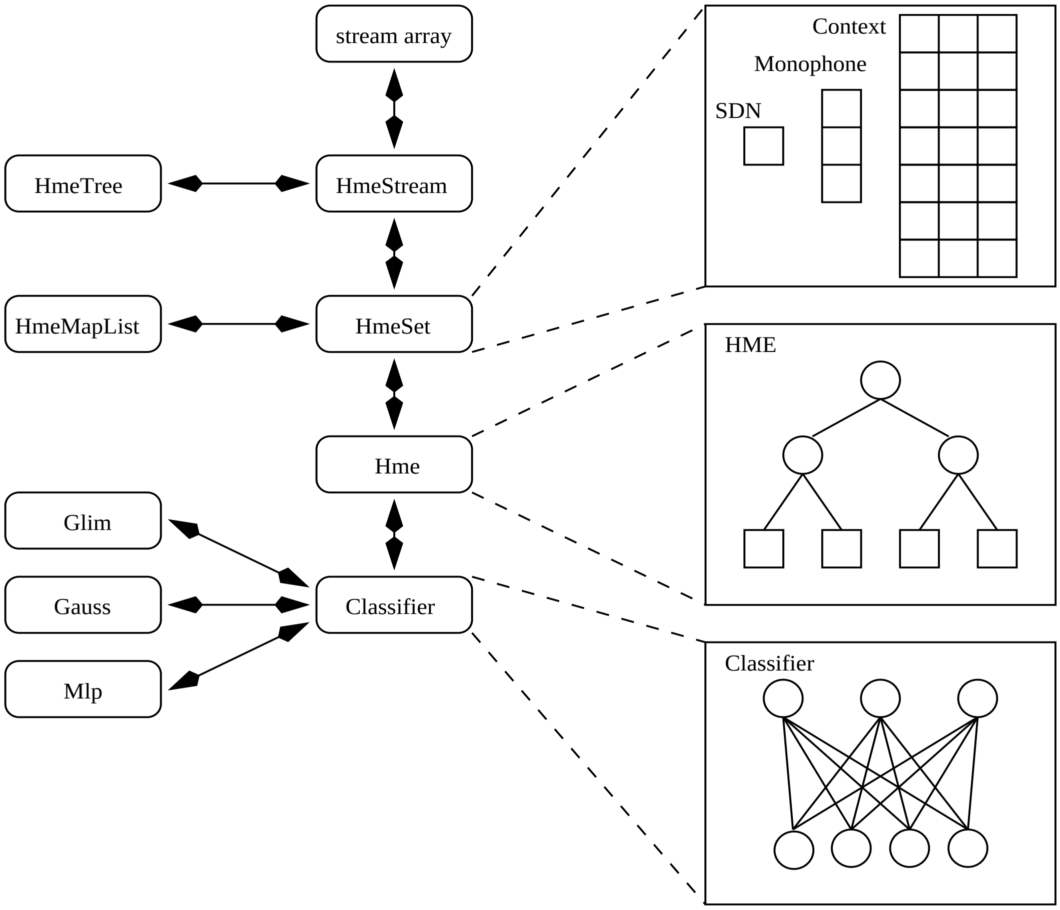

Figure 8.1: Overview: Modules of hybrid JANUS recognition system

The *HmeStream* object realizes model clustering, score computation and training by refering to the *HmeSet* object. The HmeSet object contains a set of *Hme* objects for context independent and context-dependent modeling. The HmeSet object also manages the distribution of training and testing frames to the required Hme objects. An Hme object realizes an arbitrary hierarchical mixtures of experts tree (arbitrary topology). It contains gate and expert nodes, which in turn contain *Classifier* objects. Right now, 3 types of Classifier objects are available in JANUS: Standard GLIM's as proposed for HME's by Jordan & Jacobs, Gauss classifiers necessary to build Mixtures of Gaussian experts (MGE) and two-layer perceptrons (MLP). The concept of allowing arbitrary classifiers as HME nodes generalizes the original idea of HME's which was entirely based on GLIM's. More classifier types can easily be added to JANUS, giving a great deal of flexibility to HME objects. Also, non-modular approaches like ICSI's single MLP hybrid system can be modeled by single node HME's. Apart from being used as HME nodes, all the classifier types export their functionality through the user interface, which allows to use them for other speech- or even non-speech related purposes as well.

When computing scores or updating parameters, the HmeStream refers to a *HmeTree* object to cluster phonetic contexts to model names. In the context-independent case, this decision tree is degenerated to a decision list. Once phonetic contexts are resolved to model names, the HmeStream hands them down to the HmeSet object which refers to a *HmeMapList* object to map model names to the appropriate HME and output node identifiers.

8.2 Task Description

To evaluate the system, we use the English Spontaneous Scheduling Task (ESST), a 2500 word spontaneous speech database in the domain of meeting negotiation. The database consists of roughly 8000 utterances (26 hours of speech), recorded at a sampling rate of 16 kHz. Typical examples of utterances are

```
I I MEANT MAY TWENTY SIXTH ARE YOU AVAILABLE MAY TWENTY
SIXTH BECAUSE MAY THIRTY FIRST TO JUNE THE SECOND I'LL
BE OUT OF TOWN

OKAY WE NEED TO SCHEDULE ANOTHER MEETING MY WEEK ISN'T
LOOKING THIS WEEK ISN'T LOOKING TOO BAD MONDAY I'M FREE
IN THE AFTERNOON AND TUESDAY I'M FREE IN THE MORNING SO
I GUESS WE'LL START WITH THAT AND I'LL SEE HOW YOUR
SCHEDULE IS
```

The database features lots of spontaneous effects, such as false starts, stuttering and incomplete sentences. It contains a roughly equal amount of male and female speakers. The utterances were recorded under low noise conditions using close talking headset microphones. Nevertheless, the recordings contain a considerable amount of human (coughs, breathing) and non-human (key clicks, electronic hum) noise.

8.3 General System Description

The feature space for the system is cepstrum based. ADC data is preprocessed in the following steps:

(1) Detect Speech primarily based on signal power. Use this feature to suppress non-speech segments.

(2) Compute short-time FFT over 16ms windows at a frame rate of 100 frames/sec.

(3) Convert frequency scale into a log melscale with 30 coefficients.

(4) Compute cepstrum with 13 coefficients.

(5) Compute delta and delta-delta features and merge them with cepstrum and some ADC features like power and zero crossing rate.

(6) Apply context-independent LDA and shrink the resulting 47 dimensional vector to the 32 most-significant coefficients.

(7) In some experiments, we did merge a 5-frame window of 32-dimensional features to a 160-dimensional feature to provide more context information for the networks.

Since the HME's require supervised training, we need to generate alignment paths for each training utterance, which in turn provide targets for each frame. There are many ways of computing training alignments for a connectionist system. A purely connectionist hybrid system, however, requires iterative training, where the system of a previous iteration itself is used to align the training data for the next iteration. There are two major drawbacks of this kind of training. It requires many iterations and a consistent stopping criterion, and, it relies heavily on reasonable initial network parameters. Some researchers accomplish the latter by pre-training the networks on a hand-labeled phonetic database such as TIMIT.

We use a different training scheme. Since our recognizer integrates connectionist and mixture-of-Gaussians based scoring, it is relatively easy to use a well-trained Gaussian recognizer to align the training data for the hybrid system. Therefore, we compute alignment paths for each training utterance and save them to disk. These paths are subsequently used as targets for the NN training. We found, that this training scheme worked very well, although ultimately, we might gain performance by re-training the networks on alignments that were generated by the (trained) hybrid system.

All experiments were carried out using a 3-state HMM left-right topology and 51 monophones. The resulting setup for the HmeStream therefore was as follows: 1 state discriminating HME, 3 monophone HME's and a maximum of 153 context modeling HME's for context-dependent systems.

The systems are evaluated in terms of word accuracy (WA), substitution (S), deletion (D) and insertion (I) rates, using a set of 291 test utterances which were kept apart from the training data. The number of training iterations performed and the size of the system in terms of the number of acoustic modeling parameters are reported also.

8.4 CI Systems

We trained several systems, based on different HME architectures and different HME node classifiers to evaluate the hybrid system. We started to experiment with context-independent hybrid HME systems and investigated the following architectures:

- **GLIM nodes:** Trees of depth 2 with a branching factor of 4. Gate and expert nodes were generalized linear models.
- **Gaussian nodes:** Trees of depth 1 with a branching factor of 52, which is the number of monophones in the system. The branching factor was chosen as the number of monophones to be able to use the fast initialization technique for MGE's that we presented earlier.
- **Growing trees:** Trees with a constant branching factor of 4 and GLIM nodes, adaptively grown with the constructive method presented in this thesis. The trees were grown until they contained the same number of experts (16) as the other GLIM based architecture. To speed up the tree growing phase, we used a restricted training set of about one tenth of all training utterances. However, the grown architecture was then retrained on the whole training set.
- **MLP nodes:** Trees of depth 1 with a branching factor of 4 and 2-layer MLP nodes. Each MLP contained either 100 or 300 hidden nodes. The architecture was trained by gradient ascent in log likelihood, assuming a multinomial probability model for gates and experts. Therefore, the output non-linearity of all MLP's was the softmax function.
- **Single node MLP:** HME's consisting of only one single expert node, containing a 2-layer MLP with 500 hidden nodes. This architecture is comparable to ICSI's hybrid system based on MLP's.
- **Gender dependent MLP nodes:** Separate MLP-HME's trained on male and female speakers, respectively. After training, the two gender dependent HME's were combined to a new HME, introducing an additional top-level gate. The whole architecture was then retrained for one additional iteration. This form of initalizing an HME resembles the Meta-Pi paradigm, as introduced in [18].

Results for the above systems are summarized in the following table:

System	nodes	# params	#iter	itime	WC	Subs	Dels	Ins	WA
HME-1	GLIM	421k	4	18h	66.1%	23.2%	10.7%	8.4%	**57.7%**
MGE-1	Gauss	530k	3	8h	67.8%	22.4%	9.8%	9.5%	**58.3%**
HME-2	GLIM	421k	9	7h	67.0%	22.5%	10.5%	9.1%	**57.9%**
HME-3	MLP	962k	3	26h	68.9%	21.5%	9.6%	8.1%	**60.8%**
HME-4	MLP	420k	4	17h	68.5%	21.9%	9.6%	9.3%	**59.2%**
HME-5	MLP	1.0M	3	30h	69.6%	20.6%	9.8%	7.9%	**61.7%**

In this table, *#iter* stands for the number of training iterations that were performed and *itime* stands for the amount of time required for one iteration through the training data (measured on a DEC alpha workstation). *WC, Subs, Dels, Ins* and *WA* are abbreviations for word correct rate, substitution rate, deletion rate, insertion rate and word accuracy, respectively.

We achieved the best results with systems that used MLP's as node classifiers. However this is largely due to the fact, that these systems had more parameters than the ones that were based on GLIM's. Larger GLIM based HME's have the disadvantage of increased tree traversal overhead during training and testing.

8.5 CD Systems

Next, we trained and tested context-dependent hybrid systems. Since the context-dependent posteriors are modeled by independent sets of CI and CD HME's, the context HME's can be trained separately. Also, the context HME's are trained on much smaller training sets, depending on the priors of the corresponding monophones. Therefore, the complexity of context HME's can be kept low, which is favourable both in terms of the number of additional parameters and in the additional training time. For this thesis, we trained context HME's consisting of only one expert node, a multinomial GLIM. This requires only a very modest increase in the number of parameters and in the training time. From our continuous density HMM recognizer, a polyphone clustering decision tree with 2000 context classes was available. This tree can be shrinked to any desired number of context classes. We used trees with 500 and 1000 context classes for our experiments. Training the context HME's took only about 2-5 hours and required only one iteration through the training data. After the context HME's have been trained, they were used to augment some of the context-independent hybrid systems presented in the last section. The following table summarizes the results for the context-dependent hybrid HME/HMM systems:

System	Type	CI		CD-500		CD-1000	
		WA	# param	WA	# param	WA	# param
HME-CD-1	GLIM-2-4	57.7%	421k	**60.8%**	501k	**63.8%**	581k
HME-CD-2	MLP-1-4	60.8%	962k	**61.7**	1.06M	**65.8%**	1.14M
HME-CD-3	MLP-GD	61.7%	1.0M	N/A	1.08M	**67.1%**	1.16M

The numbers reported in the *WA* columns are word accuracies. The best hybrid HME/HMM system achieved a word accuracy of 67.1% using 1000 context classes. Our context-dependent continuous-density mixture of Gaussians HMM recognizer currently achieves between 71% and 73.1% modeling 5000 context classes with tied-mixtures over 2000 distinct codebooks. This system contains over 4 million parameters, which is 4-8 times more than observed in the neural network systems, that we analyzed for this thesis.

Furthermore, decoding speed is about 2-5 times faster for the hybrid system, rendering it useful for near-realtime decoding (i.e. demo situations). Fig. 8.2 gives an overview of the performance of the various hybrid systems in terms of word accuracy.

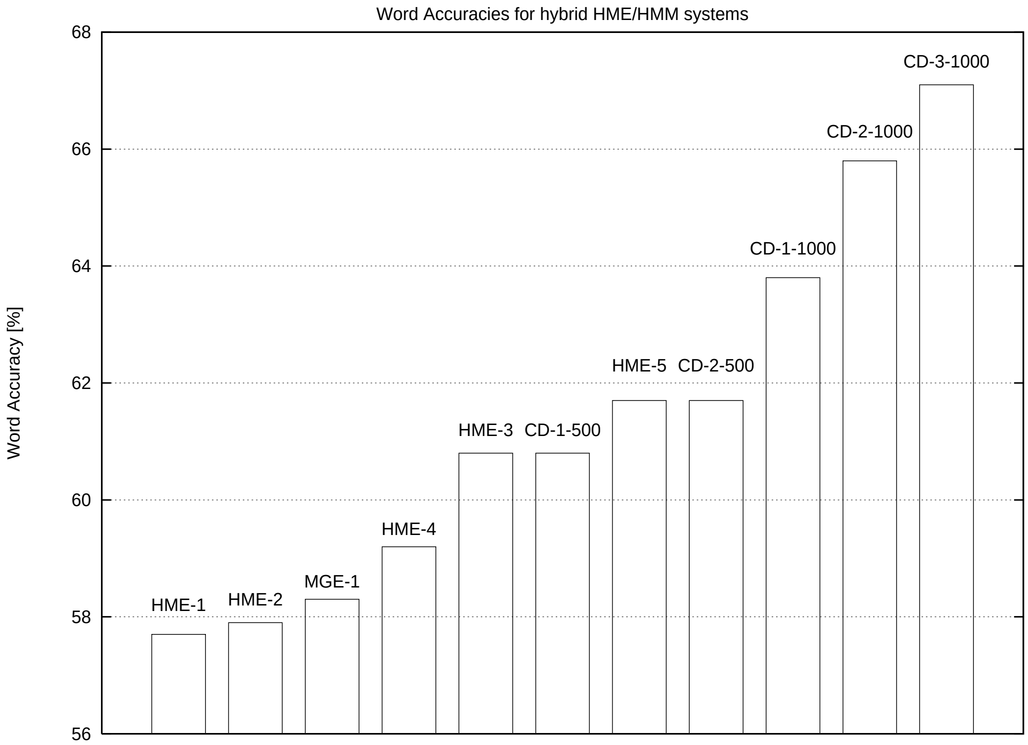

Figure 8.2: Word accuracies for several hybrid HME/HMM systems

8.6 CD Smoothing

In our context-dependent hybrid HMM system, we estimate scaled acoustic model likelihoods the following way:

$$\hat{p}(\mathbf{x}|c_j, \omega_i, s_k) = \frac{p(c_j|\omega_i, s_k, \mathbf{x})}{P(c_j|\omega_i, s_k)} \quad \frac{p(\omega_i|s_k, \mathbf{x})}{P(\omega_i|s_k)} \quad \frac{p(s_k|\mathbf{x})}{P(s_k)}$$

As in [30], we introduce a smoothing factor for the context dependent posteriors in order to compensate different dynamic ranges of context-independent and context-dependent posteriors. The above likelihood estimation is therefore modified to include a context-dependent likelihood scaling factor γ with $0.0 \leq \gamma \leq 1.0$

$$\hat{p}(\mathbf{x}|c_j, \omega_i, s_k) = \left(\frac{p(c_j|\omega_i, s_k, \mathbf{x})}{P(c_j|\omega_i, s_k)}\right)^{\gamma} \quad \frac{p(\omega_i|s_k, \mathbf{x})}{P(\omega_i|s_k)} \quad \frac{p(s_k|\mathbf{x})}{P(s_k)}$$

A smoothing factor $\gamma = 1.0$ corresponds to the original likelihood estimation, where context-dependent and context-independent scaled likelihoods are weighted equally. As γ goes towards zero, the contribution of the context-dependent HME's is reduced. For $\gamma = 0.0$ the system degenerates to a context-independent system, context-dependent likelihood estimates are fully suppressed.

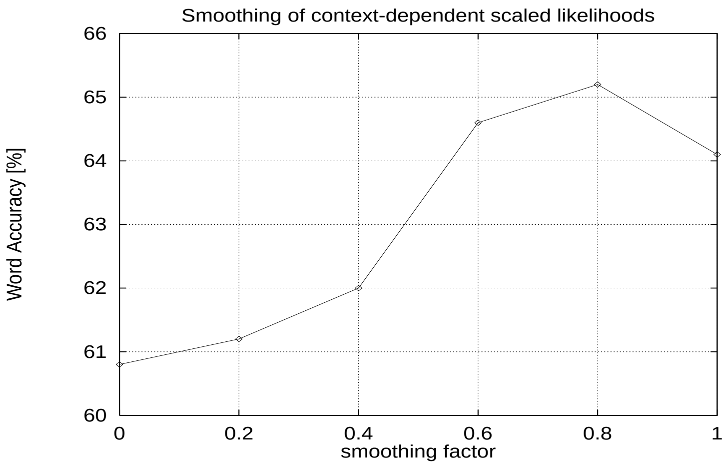

Figure 8.3: Smoothing context-dependent scaled likelihoods

The effect of this kind of smoothing can be seen in Fig. 8.3, which shows the word accuracy for different smoothing factors applied to the HME-CD-2 system.

In this experiment, the word accuracy of the system could be improved by 1.1% with a smoothing factor of 0.8. Instead of using just one single smoothing factor for all the context-dependent HME's, it might be advantageous to have separate smoothing factors for each one of the context-dependent HME's. In principle, this option is available in the current implementation of the hybrid system. However, a learning algorithm for the smoothing factors must be implemented, because they can no longer be adapted by sampling the word accuracy. This might be done in future work.

8.7 Prior Division and SDN

Our implementation of the hybrid system allows the selective activation of each single HME. This allows to experiment with different setups, without having to boot new systems from scratch. For instance, a context-dependent system can easily be switched to a context-independent one by turning off all the context networks. Furthermore, the state discriminating network (SDN) in a multi-state topology can also be switched on and off. To experimentally check the validity of theoretical results, we performed several test runs with the SDN enabled and disabled, respectively. The results were consistent with the theory for all tests. The systems with disabled SDN were always 2-3% worse than the ones with the SDN enabled, in terms of word accuracy.

Division of network outputs by class prior probabilities was observed to boost performance also. However, in some cases where we trained the networks on relatively small amounts of data, we found that prior division had the opposite effect of decreasing overall performance. Since prior probabilities are estimated by relative frequencies in the training set, smaller training set sizes will lead to poorer estimates of class priors. Especially when some of the classes have very low priors, a large training set is inevitable.

8.8 Analyzing the Systems

A hybrid speech recognition system should not only be evaluated in terms of word accuracy or word error rates. We will therefore take a closer look at some other aspects of the hybrid recognition process.

8.8.1 Sample Hypotheses

Taking a closer look at some of the recognizer's hypotheses can provide insight in the kind of errors that are made. Also, it is interesting to compare recognition hypotheses from a hybrid and a traditional system. Following is a list of typical false recognition hypotheses of the traditional HME (TRD) and the hybrid HME (HYB) system together with the correct reference (REF):

```
REF: Okay that's fine so wednesday the third at the coffee shop
TRD: We could do it so fine so wednesday the third at coffee shop
HYB: Okay that sounds fine so wednesday the third at that coffee shop

REF: should we meet again sometimes
TRD: with with should we meet again some times with
HYB: should we meet again some times

REF: Well would you be free on friday the eighth
TRD: hours now would june be you free on friday the eighth
HYB: I'm then Ron would you be free on friday the eighth

REF: okay see you then
TRD: okay see you then
HYB: okay see you then is

REF: yes today is january the fourth so yeah tomorrow is that okay
TRD: yes two days january four so yeah tomorrow is that okay
HYB: I yesterday january the four so I'm yeah tomorrow is that okay
```

Generally, both systems commit errors in the same regions. However, there are also parts, were one of the systems is detecting the right words wereas the other system is completely wrong and vice versa. This encourages the exploration of systems, where observation likelihoods are computed as a combination of neural network and parametric mixture methods.

8.8.2 Gating Probability Diagrams

One of the advantages of HME's over monolithic neural networks is the distributed way of solving the classification task. To demonstrate how the HME's that we've trained on ESST data behave in terms of gating and distributing responsibility among experts, we developed a tool that allows to plot gating probabilities (expert activations) over time for an HME. Fig. 8.4 shows such a plot for the mid-state HME of the HME-1 system presented earlier. The HME consists of 16 experts and 5 gates, organized in a 2-level tree of branching factor 4. The plot was generated by computing HME activations along a forced alignment of a recognized hypotheses. It also contains vertical lines indicating word boundaries.

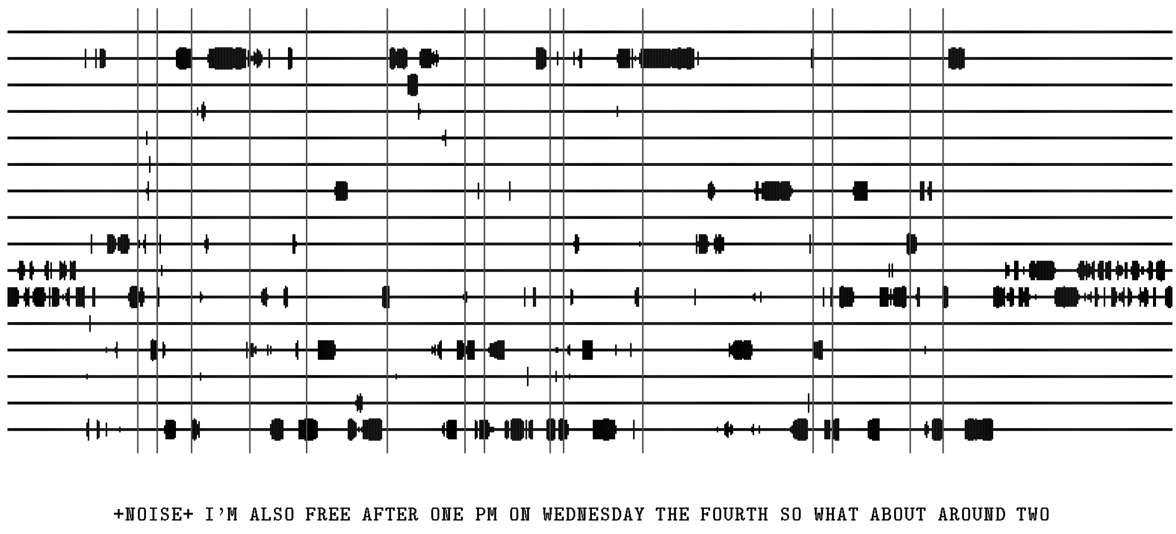

Figure 8.4: Expert activations over time for HME-2-4

The above plot reveals some interesting aspects of our hybrid HME system. The beginning and ending part of the above utterance contains long noise parts, which coincide with strong activations of just two experts (number 10 and 11 from top to bottom). Experts number 2,13 and 16 are contributing most during speech segments. There are also some experts, which are hardly ever active at all (1,6 and 8, for instance). However, we found, that in other utterances, spoken by different speakers, some of these experts show different behaviour and are contributing to the HME's decision. Nevertheless, some experts are subject to pruning, because their contribution, cummulated over a set of test utterances, is too low to be of any significance.

8.8.3 Phoneme Recognition

To analyze the frame accuracy of the hybrid recognizer, we computed monophone classification error rates and monophone confusion matrices. Since the confusion matrix for a system with 52 monophones is rather big, we decided to present a sorted list of top-5 confusions for each monophone instead. Appendix B contains such a confusion table. In the first column, it lists all monophones with their counts as measured on a list of 100 utterances. The remaining columns contain the top-5 confusion candidates, including the actual monophone itself, together with the confusion percentage.

Most confusions are consistent with what we would expect, but there are also some confusions which appear to be less obvious. The following list contains some observations regarding the confusion table:

- The phone priors are distributed highly non-uniform, some phones are very rare (for instance OY and ZH).
- The noise modeling phones (indicated with a leading +) are mostly confused among themselves. Two noise phones appear to have extremely low prior probabilities (+LA and +TH).
- The vowels are mostly confused with other vowels.
- The phone NG is often confused with the phone N.
- The phone R is often confused with AXR.
- The phones M and N are both recognized with about 60% correct rate but the phone M is much more often confused with an N than vice versa.
- The silence phone SIL is recognized with the highest accuracy (96.5%).
- The average monophone classification error rate was observed to be between 35% and 42% for the different systems.

Chapter 9

Conclusions

9.1 Summary

We developed a modular neural network based system for estimating (scaled) emission probabilities in a HMM speech recognizer. It is based on generalized hierarchical mixtures of experts (HME), allowing the integration of arbitrary neural network models into tree structured classifiers. We contributed some original work to both the field of HME's in general and the field of hybrid systems:

- We presented a constructive algorithm for HME's based on likelihood partitioning among experts. It is considerably less expensive than standard decision tree growing algorithms which require the evaluation of potential splits for all leaves.
- We investigated an alternative parameterization for both gates and experts - a mixture of Gaussian Experts (MGE). In this architecture, every node consists of a Gaussian classifier instead of the usual generalized linear model (GLIM). We showed that the MGE offers a variety of initialization techniques which allow to train it even faster than an HME.
- We developed a connectionist acoustic context modeling, based on factoring context dependent acoustic posterior probabilities. Polyphonic acoustic contexts are clustered by decision trees, which we adopt from a mixture of Gaussians based HMM recognizer. We showed, that such explicit modeling of context improves the hybrid recognizer's performance significantly.

The hybrid HMM system presented in this thesis offers many advantages over traditional mixture of Gaussians based systems. It contains considerably less parameters and allows faster decoding, especially when pruning is enabled. Furthermore, training time requirements have been reduced compared to other hybrid systems, which are based on monolithic neural networks. However, further optimizations are necessary to fully exploit the potential of this technology.

9.2 Further Work

The presented system can be enhanced in various ways. Some of the ideas that came up during the evaluation of the current system are summarized here. We believe, that the presented system still has a lot of potential for improvement. For instance, the various learning and testing parameters (especially for decoding) are most probably not yet optimal. Further work might concentrate on the following issues:

- **Mixture of likelihood estimators**
  The idea of multiple experts, whose decisions are combined by a gate can also be applied at higher levels in a speech recognizer. A hybrid system relies on discriminatively trained neural networks for (scaled) likelihood estimation whereas a traditional HMM system is based on parametric mixture densities. A system should benefit from the combination of both techniques by a gating or mediator model on top of the two (or possibly more) acoustic experts. In this case, the objective is to maximize the combined estimates of the acoustic likelihood. However, gain factors need to be applied to the different acoustic experts estimates, in order to account for the different scales.

- **Unsupervised ML adaptation**
  Unsupervised speaker adaptation has proven useful in traditional HMM speech recognizers. A (usually linear) transformation of the parameter space is iteratively updated by maximum likelihood when several utterances of a particular speaker occur. The same principle can be applied to a hybrid system. Additional front-end networks, which compute a linear transformation of the feature space can be used to account for speaker variations. Training labels for the front-end linear networks can be generated by back-propagating errors resulting from a Viterbi-alignment of decoder hypothesis. Note, that this kind of speaker adaptation can also be interpreted as a speaker adaptive LDA.

- **Improving convergence speed**
  The GEM and gradient ascent algorithms which we presented for the HME architecture are subject to lots of additional optimization techniques to improve their convergence speed. We already employed methods such as momentum terms and on-line stochastic gradients. Especially when MLP's are used as gates and experts, learning parameter optimization is crucial to reduce the number of required training iterations. Although the presented system can be trained in 2-3 days on standard workstations, a further decrease in training time is desirable.

- **Incorporating additional knowledge sources**
  The HME architecture allows in principle the use of different feature spaces for gates and experts. Why not supplying the gates with additional features such as an

estimate of the speaking rate, gender or dialect region? Together with pre-trained experts, the classification task may become easier and the whole architecture may be trainable much faster.

- **Speaker/Utterance clustering**
  Although it is a well known fact, that acoustic features are highly speaker dependent, most HMM recognizers make use of a single set of parameters for all speakers or at most, distinguish between male and female speakers. In the case of speech databases with a high degree of speaker variability, it might be more effective to cluster similar speakers into groups which then are used to train a set of neural networks. These pre-trained neural networks can then easily be integrated and trained further as HME's.

- **Learning CD smoothing factors**
  We introduced a smoothing factor between context independent and context dependent network outputs which was shown to improve performance over a non-smoothed system. We were using a single smoothing factor for all the context networks in our system. Our system also allows a separate smoothing factor for each one of the context networks. However, it remains to derive a learning algorithm for these smoothing factors (maximum likelihood). Separate smoothing factors will provide a better information scaling between the CI and CD networks.

- **Dynamic score scaling factor**
  We discovered large differences in the number of insertions and deletions among the decoded test set utterances. In some cases, the insertion rate is much higher than the deletion rate, indicating that the word insertion penalty is too low. In other cases however, the opposite behaviour can be observed (for the same language model parameters). It seems, that the variation in the acoustic scores leads to different relative weights of the language model parameters. An adaptive score scaling factor might help to overcome this effect.

- **Confidence measure based on posteriors**
  Since the acoustic models in a hybrid system are trained discriminatively, it might be useful to derive a phone or word confidence measure based on the networks estimates of frame posteriors. Furthermore, a simple measure of the frame confidence (such as the difference in score between the best and the second best acoustic model) might be useful to dynamically adjust the search beam during decoding.

Appendix A

Question Set for Decision Trees

Question-Name	Set of Phonemes covered
NOISES	+BR +HU +NH +SM +TH +LA
HUMAN-NOISES	+BR +HU +SM +TH +LA
LAUGHTER	+LA
UHHUH	+F
SILENCES	SIL
CONSONANT	P B F V TH DH T D S Z SH ZH CH JH K G HH M N NG R Y W L ER DX AXR
CONSONANTAL	P B F V TH DH T D S Z SH ZH CH JH K G HH M N NG DX
OBSTRUENT	P B F V TH DH T D S Z SH ZH CH JH K G
SONORANT	M N NG R Y W L ER AXR DX
SYLLABIC	AY OY EY IY AW OW EH IH AO AE AA AH UW UH IX AX ER AXR
VOWEL	AY OY EY IY AW OW EH IH AO AE AA AH UW UH IX AX
DIPHTHONG	AY OY EY AW OW
CARDVOWEL	IY IH EH AE AA AH AO UH UW IX AX
VOICED	B D G JH V DH Z ZH M N NG W R Y L ER AY OY EY IY AW OW EH IH AO AE AA AH UW UH DX AXR IX AX
UNVOICED	P F TH T S SH CH K
CONTINUANT	F TH S SH V DH Z ZH W R Y L ER
DEL-REL	CH JH
LATERAL	L
ANTERIOR	P T B D F TH S SH V DH Z ZH M N W Y L DX
CORONAL	T D CH JH TH S SH DH Z ZH N L R DX
APICAL	T D N DX
HIGH-CONS	K G NG W Y
BACK-CONS	K G NG W
LABIALIZED	R W ER AXR
STRIDENT	CH JH F S SH V Z ZH

Question-Name	Set of Phonemes covered
SIBILANT	S SH Z ZH CH JH
BILABIAL	P B M W
LABIODENTAL	F V
LABIAL	P B M W F V
INTERDENTAL	TH DH
ALVEOLAR-RIDGE	T D N S Z L DX
ALVEOPALATAL	SH ZH CH JH
ALVEOLAR	T D N S Z L SH ZH CH JH DX
RETROFLEX	R ER AXR
PALATAL	Y
VELAR	K G NG W
GLOTTAL	HH
ASPIRATED	HH
STOP	P B T D K G M N NG
PLOSIVE	P B T D K G
FLAP	DX
NASAL	M N NG
FRICATIVE	F V TH DH S Z SH ZH HH
AFFRICATE	CH JH
APPROXIMANT	R L Y W
LAB-PL	P B
ALV-PL	T D
VEL-PL	K G
VLS-PL	P T K
VCD-PL	B D G
LAB-FR	F V
DNT-FR	TH DH
ALV-FR	SH ZH
VLS-FR	F TH SH
VCD-FR	V DH ZH
ROUND	AO OW UH UW OY AW OW
HIGH-VOW	IY IH UH UW IX
MID-VOW	EH AH AX
LOW-VOW	AA AE AO
FRONT-VOW	IY IH EH AE
CENTRAL-VOW	AH AX IX
BACK-VOW	AA AO UH UW
TENSE-VOW	IY UW AE
LAX-VOW	IH AA EH AH UH
ROUND-VOW	AO UH UW
REDUCED-VOW	IX AX
REDUCED-CON	AXR

Question-Name	Set of Phonemes covered
REDUCED	IX AX AXR
LH-DIP	AY AW
MH-DIP	OY OW EY
BF-DIP	AY OY AW OW
Y-DIP	AY OY EY
W-DIP	AW OW
ROUND-DIP	OY AW OW
LIQUID-GLIDE	L R W Y
W-GLIDE	UW AW OW W
LIQUID	L R
LW	L W
Y-GLIDE	IY AY EY OY Y
LQGL-BACK	L R W

Appendix B

Monophone Confusion Table

Correct Phone	1	2	3	4	5
+BR,14988	+BR (53.776%)	+SM (14.905%)	SIL (7.986%)	+NH (7.686%)	+F (5.171%)
+F,9129	+F (52.788%)	+BR (12.794%)	AY (4.962%)	+HU (3.626%)	N (3.494%)
+HU,1710	SIL (11.754%)	+BR (11.345%)	+F (9.240%)	+NH (8.538%)	+HU (8.129%)
+LA,0					
+NH,27337	+NH (70.728%)	SIL (11.793%)	+SM (6.566%)	+BR (6.113%)	+HU (0.647%)
+SM,11179	+SM (63.360%)	+BR (15.556%)	+NH (12.228%)	SIL (5.314%)	F (1.565%)
+TH,0					
AA,1704	AA (30.927%)	AO (12.617%)	AY (10.915%)	+F (10.622%)	AW (5.869%)
AE,4248	AE (48.682%)	AY (8.781%)	EY (6.097%)	EH (5.508%)	IH (4.355%)
AH,4396	AH (35.873%)	IY (7.302%)	+F (6.938%)	AX (5.823%)	AY (4.504%)
AO,4057	AO (53.882%)	AY (7.247%)	OW (6.655%)	L (4.757%)	+F (4.585%)
AW,3058	AW (34.565%)	AY (11.772%)	+F (10.203%)	OW (9.287%)	AO (8.600%)
AX,4290	AX (21.655%)	IH (9.207%)	+F (5.991%)	AH (5.921%)	AE (5.524%)
AXR,1533	R (33.203%)	AXR (26.419%)	ER (8.089%)	UW (6.393%)	AX (3.196%)
AY,8195	AY (59.890%)	+F (7.468%)	AO (4.454%)	AE (3.563%)	EY (3.405%)
B,2747	B (68.657%)	DH (3.932%)	D (3.640%)	M (2.803%)	P (2.039%)
CH,506	CH (43.281%)	S (9.091%)	JH (8.696%)	T (8.300%)	K (7.510%)
D,4708	D (55.501%)	N (6.967%)	T (6.670%)	B (3.951%)	DH (3.717%)
DH,4463	DH (58.122%)	D (8.582%)	B (4.549%)	N (3.742%)	TH (3.495%)
DX,687	DX (38.428%)	T (15.429%)	IY (6.114%)	D (5.677%)	B (4.076%)
EH,5590	EH (40.626%)	AE (12.701%)	AY (9.911%)	R (4.991%)	IH (4.347%)
ER,3703	ER (63.003%)	R (24.764%)	AXR (2.457%)	AY (1.728%)	EY (1.512%)
EY,5590	EY (71.521%)	IY (10.877%)	IH (2.934%)	EH (1.932%)	AE (1.538%)
F,5490	F (84.390%)	TH (4.645%)	+NH (1.949%)	+BR (1.712%)	+SM (1.202%)
G,1441	G (46.495%)	K (13.393%)	D (11.797%)	B (3.747%)	N (3.400%)
HH,1927	HH (50.337%)	+BR (12.818%)	+NH (7.317%)	AY (5.138%)	AE (2.750%)
IH,2637	IH (39.477%)	EY (7.433%)	UW (6.560%)	AH (5.650%)	AX (3.906%)
IX,1811	IX (29.376%)	IY (17.449%)	IH (9.442%)	EY (9.277%)	AX (4.252%)
IY,10362	IY (74.551%)	EY (9.303%)	Y (2.422%)	UW (1.776%)	IX (1.756%)
JH,622	JH (46.463%)	T (15.756%)	D (6.752%)	CH (4.502%)	K (3.698%)
K,5774	K (74.645%)	T (6.304%)	+SM (3.395%)	SIL (3.135%)	+BR (2.598%)
L,5621	L (57.161%)	+F (8.611%)	OW (6.138%)	AO (2.597%)	R (2.491%)
M,6345	M (60.772%)	N (19.196%)	+BR (3.830%)	+F (3.515%)	W (1.481%)
N,13024	N (63.506%)	M (6.741%)	+BR (5.413%)	+F (3.209%)	NG (2.434%)
NG,1751	NG (34.609%)	N (29.640%)	M (5.768%)	IY (4.797%)	EY (4.169%)
OW,4039	OW (38.623%)	+F (9.557%)	L (7.923%)	AO (5.620%)	AY (4.085%)
OY,20	EH (45.000%)	AY (30.000%)	AO (10.000%)	OY (5.000%)	OY (0.000%)
P,1221	P (44.062%)	K (16.298%)	T (7.043%)	B (6.470%)	SIL (4.586%)
R,5334	R (69.835%)	ER (6.580%)	AY (4.124%)	AXR (2.081%)	+F (1.669%)

Correct Phone	1	2	3	4	5
S,8708	S (83.532%)	Z (6.431%)	F (2.664%)	T (2.251%)	TH (2.079%)
SH,597	SH (53.266%)	S (13.903%)	CH (8.878%)	T (7.873%)	Z (6.365%)
SIL,53729	SIL (96.516%)	+BR (1.539%)	+SM (0.631%)	+NH (0.281%)	K (0.143%)
T,13190	T (61.266%)	K (7.779%)	S (3.215%)	D (2.926%)	+SM (2.449%)
TH,4353	TH (53.986%)	F (10.705%)	+BR (4.663%)	SIL (4.319%)	+SM (4.273%)
UH,1147	UH (40.977%)	UW (8.980%)	EY (8.195%)	IH (7.934%)	AX (6.103%)
UW,4354	UW (52.526%)	IY (5.122%)	EY (5.076%)	IH (4.410%)	+BR (2.802%)
V,2565	V (42.105%)	F (16.725%)	M (4.405%)	B (3.782%)	N (3.197%)
W,5849	W (69.345%)	L (5.779%)	AO (4.360%)	R (4.274%)	M (1.351%)
Y,1755	Y (47.008%)	IY (28.262%)	UW (3.305%)	IH (2.963%)	EY (2.336%)
Z,2994	Z (59.987%)	S (29.125%)	T (2.204%)	DH (1.035%)	TH (0.969%)
ZH,6	IY (50.000%)	F (33.333%)	+BR (16.667%)	+BR (0.000%)	+BR (0.000%)

Bibliography

[1] Bengio, Y., De Mori, R., Flammia, G. & Kompe, R. (1992) *Global optimization of a neural network – Hidden Markov Model Hybrid.* IEEE Trans. on Neural Networks, Vol. 3, No. 2, 252-259, 1992.

[2] Bourlard, H., Konig, Y., Morgan, N. (1994) *REMAP: recursive estimation and maximization of a posteriori probabilities – Application to transition-based connectionist speech recognition.* Technical Report, International Computer Science Institute, ICSI TR-94-064, 1994.

[3] Bourlard, H., Morgan, N. (1994) *Connectionist Speech Recognition – A Hybrid Approach.* Kluwer Academic Press, 1994.

[4] Bourlard, H., Morgan, N. (1993) *Continuous Speech Recognition by Connectionist Statistical Methods.* IEEE Trans. on Neural Networks, Vol. 4, No. 6, Nov. 1993.

[5] Bourlard, H., Morgan, N. (1992) *A context dependent neural network for continuous speech recognition.* IEEE Proc. Intl. Conf. on Acoustics, Speech and Signal Processing, II:349-352, San Francisco, CA.

[6] Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984) *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA.

[7] Bridle, J. (1989) *Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition.* In Neurocomputing: Algorithms, Architectures, and Applications, F. Fogelman-Soulie and J. Hérault, eds. Springer Verlag, New York.

[8] Buntine, W. (1991) *Learning classification trees.* NASA Ames Research Center Tech. Rep. FIA-90-12-19-01, Moffett Field, CA.

[9] Cohen, M., Murveit, H., Bernstein, J., Price, P. & Weintraub, M. (1990) *The DECIPHER speech recognition system.* in Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing, 77-80, Albuquerque, NM.

[10] Dempster, A. P., Laird, N. M. & Rubin D. B. (1977) *Maximum likelihood from incomplete data via the EM algorithm.* J. R. Statist. Soc. B 39, 1-38.

[11] Duda, R. O. & Hart, P. E. (1973) *Pattern Classification and Scene Analysis.* John Wiley, New York.

[12] Finney, D. J. (1973) *Statistical Methods in Biological Assay.* Hafner, New York.

[13] Franco, H., Cohen, M., Morgan, N., Rumelhart, D. & Abrash V. (1994) *Context-dependent connectionist probability estimation in a hybrid Hidden Markov Model – Neural Net speech recognition system.* Computer Speech and Language, Vol. 8, No 3, 211-222, July 1994.

[14] Franzini, M., Lee, K. F., Waibel, A. (1990) *Connectionist Viterbi Training: a new hybrid method for continuous speech recognition.* IEEE Proc. Intl. Conf. on Acoustics, Speech and Signal Processing, 425-428, Albuquerque, NM, 1990.

[15] Friedman, J. H. (1991) *Multivariate adaptive regression splines.* Ann. Statistics 19, 1-141.

[16] Fritsch, J. & Rogina, I. (1996) *The Bucket Box Intersection (BBI) Algorithm for Fast Approximative Evaluation of Diagonal Mixture Gaussians* IEEE Intl. Conf. on Acoustics, Speeech and Signal Processing, May 1996, Atlanta, GA.

[17] Gish, H. (1990) *A probabilistic approach to the understanding and training of neural network classifiers.* Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing 1361-1364, Albuquerque, NM.

[18] Hampshire II, J. B., Waibel A. H. (1989) *The Meta-Pi Network: Building Distributed Knowledge Representations for Robust Pattern Recognition* Tech. Rep. CMU-CS-89-166, Carnegie Mellon University, Pittsburgh PA, August 1989.

[19] Haykin, S. (1991) *Adaptive Filter Theory.* Prentice-Hall, Englewood Cliffs, NJ.

[20] Haykin, S. (1994) *Neural networks: a comprehensive foundation.* Macmillan, New York.

[21] Hochberg, M. M., Cook, G. D., Renals, S. J., Robinson, A. J. & Schechtman, R. S. (1995) *The 1994 ABBOT Hybrid Connectionist-HMM Large-Vocabulary Recognition System.* In Spoken Language Systems Technology Workshop, 170-176, ARPA, Jan. 1995.

[22] Jacobs, R. A., Jordan, M. I., Nowlan, S. J. & Hinton, G. E. (1991) *Adaptive mixtures of local experts.* Neural Comp. 3 , 79-87.

[23] Jelinek, F. (1976) *Continuous speech recognition by statistical methods.* Proc. of IEEE, Vol. 64, No. 4, 532-555.

[24] Jelinek, F. (1990) *Self-organized modelling for speech recognition.* In Readings in Speech Recognition, A. Waibel and K. F. Lee (eds), 450-503, Morgan Kaufmann, 1990.

[25] Jiang, L., Barnard, E. (1994) *Choosing contexts for neural networks.* Oregon Graduate Institute Technial Report, 1994.

[26] Jordan, M. I., Jacobs, R. A. (1992) *Hierarchies of adaptive experts.* In Advances in Neural Information Processing Systems 4, J. Moody, S. Hanson & R. Lippmann, eds., pp. 985-993. Morgan Kaufmann, San Mateo, CA.

[27] Jordan, M. I., Jacobs, R. A. (1994) *Hierarchical Mixtures of Experts and the EM algorithm.*, Neural Computation 6, 181-214, MIT Press.

[28] Jordan, M. I., Xu, L. (1993) *Convergence Properties of the EM Approach to Learning in Mixture-of-Experts Architectures.* Computational Cognitive Science Tech. Rep. 9301, MIT, Cambridge, MA.

[29] Jordan, M. I., Jacobs, R. A. (1996) *Modular and hierarchical learning systems.* In press: M. Arbib, ed., The Handbook of Brain Theory and Neural Networks. Cambridge, MA.

[30] Kershaw, D. J., Hochberg, M. M. & Robinson, A. J. (1995) *Context-Dependent Classes in a Hybrid Recurrent Network-HMM Speech Recognition System.* Tech. Rep. CUED/F-INFENG/TR217, Cambridge University Engineering Department, Cambridge, England.

[31] Kershaw, D. J., Robinson, T., Hochberg, M. (1995) *Context-Dependent Classes in a Hybrid Recurrent Network-HMM Speech Recognition System.* In Advances in Neural Information Processing Systems 8.

[32] Lee, K. F. (1988) *Large vocabulary, speaker-independent continuous speech recognition: The SPHINX system.* Kluwer Academic Publishers, 1988.

[33] Lippman, R. P. (1989) *Review of neural networks for speech recognition.* Neural Computation, Vol. 1, No. 1, 1-38, 1989.

[34] McCullagh, P., Nelder, J. A. (1983) *Generalized Linear Models.* Chapman and Hall, London.

[35] Moody, J. (1989) *Fast learning in multi-resolution hierarchies.* In Advances in Neural Information Processing Systems, D.S. Touretzky, ed. Morgan Kaufmann, San Mateo, CA.

[36] Moody, J., Darken, C. J. (1989) *Fast learning in networks of locally tuned processing units.* Neural computation 1, 281-294.

[37] Morgan, N., Bourlard, H. (1992) *Factoring Networks by a Statistical Method.* Neural Computation, Vol 4, No. 6, 835-838, 1992.

[38] Morgan, N., Bourlard, H. (1995) *An Introduction to Hybrid HMM/Connectionist Continuous Speech Recognition.* Signal Processing Magazine, 25-42, May 1995.

[39] Morgan, N. (1994) *Big Dumb Neural Nets (BDNN): a working brute force approach to speech recognition.* Proc. of the ICNN, Vol. 7, 4462-4465, 1994.

[40] Nowlan, S. J. (1990) *Maximum likelihood competitive learning.* In Advances in Neural Information Processing Systems 2, D.S. Touretzky, ed. Morgan Kaufmann, San Mateo, CA.

[41] Nowlan, S. J. (1991) *Soft Competitive Adaptation: Neural Network Learning Algorithms Based on Fitting Statistical Mixtures.* Tech. Rep. CMU-CS-91-126, CMU, Pittsburgh, PA.

[42] Peterson, G. E., Barney, H. (1952) *Control methods used in a study of the vowels.* Journal Acoust. Soc. America 24, 175-184.

[43] Proceedings of LVCSR Hub 5 workshop, Apr. 29 - May 1 (1996) MITAGS, Linthicum Heights, Maryland.

[44] Quinlan, J. R. (1986) *Induction of decision trees.* Machine Learn. 1, 81-106.

[45] Quinlan, J. R., Rivest, R. L. (1989) *Inferring decision trees using the Minimum Description Length Principle.* Information and Computation 80, 227-248.

[46] Richard, M. D., Lippman, R. P. (1991) *Neural network classifiers estimate Bayesian a posteriori probabilities.* Neural Computation 3, 461-483, 1991.

[47] Ripley, B. D. (1993) *Statistical Aspects of Neural Networks.* In Barndorff-Nielsen, O. E., Jensen, J. L. & Kendall, W. S. (eds) Networks and Chaos: Statistical and Probabilistic Aspects, London: Chapman & Hall.

[48] Sarle, W. S. (1994) *Neural Networks and Statistical Models.* In Proc. Nineteenth Annual SAS Users Group Internat. Conference, April 1994.

[49] Strömberg, J. E., Zrida, J. & Isaksson, A. (1991) *Neural trees – using neural nets in a tree classifier structure.* IEEE Internat. Conf. on Acoustics, Speech and Signal Proc., 137-140.

[50] Syrdal, A. K., Gopal, H. S. (1986) *A perceptual model of vowel recognition based on the auditory representation of American English vowels.* Journal Acoust. Soc. America 79(4) 1086-1100

[51] Tebelskis, J. (1995) *Speech Recognition using Neural Networks* Ph. D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh PA.

[52] Waibel, A., Lee, K. F. (eds) (1990) *Readings in Speech Recognition.* Morgan Kaufmann 1990.

[53] Waibel, A., Finke, M., Gates, D., Gavalda, M., Kemp, T., Lavie, A., Levin, L., Maier, M., Mayfield, L., McNair, A., Rogina, I., Shima, A., Sloboda, T., Woszczyna, M., Zeppenfeld, T., Zhan, P. (1996) *JANUS-II - Advances in Spontaneous Speech Translation.* Internat. Conf. Acoustics, Speech and Signal Proc., May 1996, Atlanta, Georgia.

[54] Waterhouse, S. R. (1993) *Speech Recognition using Hierarchical Mixtures of Experts.* M. Phil. Thesis, University of Cambridge, England.

[55] Waterhouse, S. R., Robinson, A. J. (1994) *Classification using Hierarchical Mixtures of Experts.* IEEE Workshop on Neural Networks for Signal Processing IV, 177-186.

[56] Waterhouse, S. R., Robinson, A. J. (1995) *Constructive Algorithms for Hierachical Mixtures of Experts.* In Advances in Neural Information Processing Systems 8.

[57] Xu, L., Jordan, M. I. & Hinton, G. E. (1995) *An Alternative Model for Mixtures of Experts.* In Advances in Neural Information Processing Systems 8.

[58] Zavaliagkos, G., Zhao, Y., Schwartz, R. & Makhoul, J. *A Hybrid Segmental Neural Net/Hidden Markov Model System for Continuous Speech Recognition.*, IEEE Trans. on Speech and Audio Processing, Vol. 2, No. 1, 151-160, 1994.

[59] Zhao, Y., Schwartz, R., Sroka, J. & Makhoul, J. (1995) *Hierarchical Mixtures of Experts Methodology Applied to Continuous Speech Recognition.* Internat. Conf. Acoustics Speech and Signal Proc., Vol 5, 3443-3446, May 1995.