Karlsruhe Institute of Technology

GMM free ASR using DNN based Cluster Trees

Bachelor Thesis of

Linchen Zhu

At the Department of Informatics
Institute of Anthropomatics and Robotics (IAR)

Reviewer:	Prof. Alexander Waibel
Second reviewer:	Dr. Sebastian Stueker
Advisor:	Dipl.-Inform. Kevin Kilgour
Second advisor:	Prof. Ian Lane

Duration: 15th October 2014 – 14th February 2015

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 14.02.2015

.....

(**Linchen Zhu**)

Abstract

Deep neural networks (DNN), as a kind of statistical models, have shown their superiority over Gaussian Mixture Models (GMM) in acoustic modeling when they are used to estimate the emission distributions of HMM states. However a standard DNN-HMM hybrid system is still reliant on GMMs in two aspects - initial training alignments and cluster tree building. Recent work has shown that the training of a context independent DNN can be flat started without the initial alignments generated by a trained GMM-HMM system. In this work we propose a novel GMM-free approach for phonetic cluster tree building.

To do this a context independent DNN is trained first. From this the average CI-DNN output for each polyphone state is calculated by passing the training samples through the CI-DNN, the results from which can then be used to measure the entropy distance between two sets of polyphone states in the clustering process.

Experiments are performed to realize our novel approach, and in addition, cluster trees of different sizes are built to investigate the best performance of our novel approach. All implemented systems are tested on a test database, the test results show that DNN based cluster trees outperform GMM based trees when the number of leaves is greater than 12k, and that both approaches achieve their best performance at 18k leaves.

Contents

1. Introduction

1.1 Motivation

In recent decades the recognition accuracy of automatic speech recognition (ASR) systems has greatly improved, a considerable part of these improvements is due to the technologies developed in the field of acoustic modeling. Most state-of-the-art ASR systems take advantage of statistically-based acoustic models, which means the parameters of acoustic models are estimated from a large amount of training data with the help of statistical methods. The two most widely used statistical approaches to acoustic modeling are the conventional Gaussian mixture model (GMM) based hidden Markov model (HMM), also known as GMM-HMM acoustic model, and a combination of deep neural network (DNN) and HMM which is familiar as the DNN-HMM hybrid system.

GMM-HMM systems have long been proven to be successful and are widely used as a result of their numerous advantages, including the fact that GMMs have fewer parameters compared with other statistical models, the training of GMM-HMM models can be easily parallelized, and the performance of these models can be further improved with speaker adaptation training. Despite this, the GMM based approach still has drawbacks, for example, it assumes a GMM distribution of the acoustic feature space; however this assumption may not be true for speech data. One of the major drawbacks to using GMM-HMM models is that they are generative models, which in classification tasks, are routinely outperformed by discriminative models. These drawbacks to the GMM based approach can be overcome by introducing DNNs to the acoustic modeling, since DNNs trained with new deep learning methods have made notable advances in speech recognition and outperformed GMM-HMM systems on various large vocabulary continuous speech recognition (LVCSR) tasks. Nevertheless, state-of-the-art DNN-HMM hybrid systems still rely on GMMs in two areas - initial training alignments and cluster tree building. Recent studies have shown that DNN training can be flat started without the initial alignment generated by a well-trained GMM-HMM system [SHBL14] [ZW14], but almost no previous work has been done to prove that cluster trees can be built without using GMMs. Therefore this work proposes a novel GMM-free approach for cluster tree building, and experiments are carried out to realize this novel approach.

1.2 Contribution

To our knowledge, almost all existing cluster tree building approaches are based on GMMs or single Gaussians, in this work we propose a novel DNN based approach to building clus-

ter trees and experiments show that the DNN based approach achieves better performance than the conventional GMM based approach. With the help of our novel cluster tree building approach, a totally GMM-free ASR system with state-of-the-art recognition accuracy can be built, which will simplify the code library of an ASR system since the GMM library is no longer required for the implementation. Furthermore, that DNNs have been successfully applied to the building of cluster trees shows that they still have considerable further potential, and researchers struggling for new interests would do well to consider investigating the potential uses and modeling power of DNNs in greater detail.

1.3 Layout

The next two chapters of this thesis provide the background knowledge to our work. Chapter two first gives a brief overview of a typical ASR, followed by a detailed description of acoustic modeling, since this work mainly concerns acoustic modeling in ASR. An introduction to artificial neural networks can be found in chapter three, with the burgeoning subfield of deep learning also introduced in this chapter. Chapter four analyzes the problem of cluster tree building, and describes the previous work on the semi-continuous HMM to solve this problem, before introducing our novel DNN based approach. Chapter five demonstrates the pipeline to build the DNN based cluster trees and also the steps followed to test their performance. The experiments performed to implement the novel approach and baseline systems are documented in the sixth chapter, as well as an evaluation of the systems implemented. The last chapter discusses the most relevant conclusions of this work and provides some suggestions for further work.

2. Acoustic Modeling

This chapter first gives a broad overview of a typical ASR system, after which the main components of an ASR system are briefly discussed. The remaining part of this chapter presents more detailed background knowledge concerning acoustic modeling, covering the conventional GMM-HMM acoustic model and the standard approach of context dependent acoustic modeling.

2.1 Overview of ASR

The input of an ASR system is an analog speech signal, the task of the ASR system is to find the most likely word sequence $\hat{W}$ that matches the input speech signal, namely:

$$\hat{W} = \arg\max_{W} P(W|A),$$

where A is a sequence of acoustic feature vectors extracted from the analog speech signal. After applying Bayes' rule the fundamental formula of speech recognition is obtained, and in this optimization problem $P(A)$ is a constant which can be ignored, thus the formula can be further simplified:

$$\begin{aligned} \hat{W} &= \arg\max_{W} \frac{P(W)\,P(A|W)}{P(A)} \\ &= \arg\max_{W} P(W)\,P(A|W). \end{aligned}$$

Typically an ASR system consists of the components shown in figure 2.1. The following sections briefly discuss the four most important components of the ASR system, which are preprocessing, language model, acoustic model and decoder respectively.

2.1.1 Preprocessing and Feature Extraction

The input for an ASR system is the analog speech signal, the aim of preprocessing is to extract feature vector sequences which contain the most important information from the original speech signal for further analysis. The analog speech signal is first converted into a sequence of digital signals which can then be processed by computers after sampling and A/D conversion.

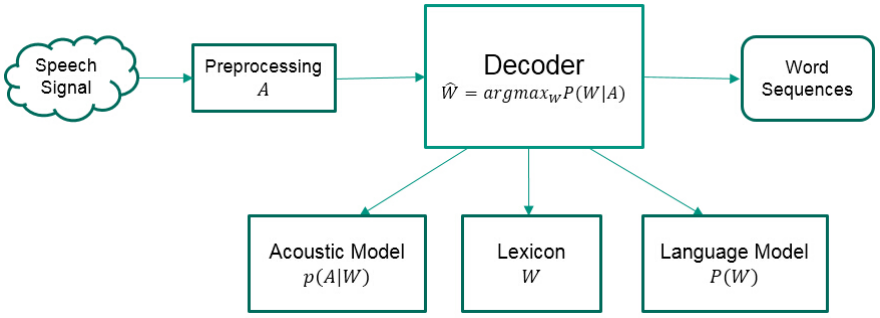

Figure 2.1: *Components of a typical ASR system, the input of an ASR system is the speech signal and the output is the best matched word sequence.*

The speech signal is non-stationary as it varies rapidly with time, however in a short time period of 10-40ms it is considered to be stationary. Therefore a sequence of consecutive digital signals is split into frames covering a short time period, which is normally 32ms. Each frame partly overlaps the neighbouring frames, this is implemented by shifting the frame window several milliseconds to the right each time - this step is also known as framing.

A time domain speech signal cannot be used directly by an ASR system, it is essential to extract the information in a speech signal in the frequency domain which contains the relevant features of speech. As speech signal in a frame is assumed to be stationary, a series of feature extraction algorithms are applied to the speech frames, before finally a feature vector for each frame is extracted. Some of the most commonly used feature coefficients are log-MEL, MFCC, and PLP.

After preprocessing the analog speech signal is converted to a sequence of n-dimensional vectors, called the observation sequence or feature vector sequence, where n is the dimension of the feature vectors. These feature vector sequences are then utilized for acoustic modeling to train acoustic models.

2.1.2 Language Model

Language models are used to estimate the a priori probability of a word sequence W appearing in the natural language, namely $P(W)$ in the fundamental formula of speech recognition. Currently there are two main approaches for language modeling, one is the n-gram language model, the other is the neural network language model. The a priori probability of a word sequence $w_1, ..., w_k$ can be calculated using chain rule as:

$$P(w_1, ..., w_k) = P(w_1) P(w_2|w_1) ...P(w_k|w_1, ..., w_{k-1}) .$$

Thus the main task of the n-gram language model is to estimate the a posteriori probability of a word w_k given an arbitrary word sequence $w_1, ..., w_{k-1}$, which is also called history, namely $P(w_k|w_1, ..., w_{k-1})$. However this idea is not feasible, because a large number of histories can not be seen in the training corpus. Thus when the n-gram model is employed as the language model, an assumption must be made to simplify the modeling:

$$P(w_k|w_1, ..., w_{k-1}) \approx P(w_k|w_{k-n+1}, ..., w_{k-1}) ,$$

which means the probability of next word depends only on the n-1 most recent words, this is also known as the Markov assumption. Normally maximum likelihood estimation (MLE) is used to approximate the probabilities of n-gram model:

$$P\left(w_k|w_{k-n+1},...,w_{k-1}\right) = \frac{C\left(w_{k-n+1},...,w_{k-1},w_k\right)}{C\left(w_{k-n+1},...,w_{k-1}\right)},$$

where $C(W)$ denotes the count of word sequence W appearing in the training corpus.

2.1.3 Acoustic Model

Acoustic models are used to estimate the conditional probability $P(A|W)$, which is the a posteriori probability of the feature vector sequence A when the hypothesized word sequence W is uttered. Human speech is composed of a sequence of acoustic units; there are different kinds of acoustic units at different levels including phonemes, words, phrases and sentences. Informally, the task of acoustic modeling is to model the relation between the observed feature vectors and acoustic units, which means that acoustic models should know what these acoustic units sound like. Statistically-based acoustic modeling builds models for each acoustic unit and trains the models on the corresponding speech data.

If the modeled acoustic units are words or sentences, theoretically different acoustic units can be better discriminated from each other. However this will result in an excessive number models and parameters which cannot be handled efficiently by computers. Moreover a great number of acoustic units will have insufficient training data for a robust estimation of model parameters or even cannot be seen in the training corpus. As a trade-off, phonemes are usually the modeled acoustic units of acoustic modeling. Phonemes are the smallest components of a language, usually a language has approximately fifty different phonemes, and thus there is enough training data for each phoneme.

In order to take varying speech rates into consideration, most ASR systems employ the hidden Markov model (HMM) [RJ86] for acoustic modeling. A combination of HMM and GMM (GMM-HMM) is one of most successful acoustic models and as such plays a dominant role in ASR. However recent research has shown that employing DNNs in acoustic modeling can greatly improve recognition accuracy, and thus application of DNNs in ASR will receive greater attention in the future.

2.1.4 Decoder

The task of a decoder is to find the most likely word sequence using the knowledge of acoustic models, language models and lexicon. Typically the search space is huge; for example, to search a sentence with N words in a vocabulary with size M, there can be M^N possible word sequences. Thus a brute force approach is not feasible, and the decoder normally reduces the search space with the Viterbi algorithm [Vit67] based on the principle of dynamic programming. Additional methods such as beam search [Low76], or weighted finite state transducer approach [Moh04] are generally still needed to further accelerate the search. Unlike the Viterbi algorithm which surely finds the best word sequence, beam search prunes paths that are much worse than the current best path while searching, however the best path may also be pruned.

2.2 GMM-HMM Acoustic Model

The GMM-HMM acoustic model is a combination of hidden Markov model and Gaussian mixture model. It is one of the most successful acoustic models in modern speech recognition systems, the theory behind GMM-HMM models is well studied and a range of algorithms are proposed to further improve the performance of GMM-HMM models, including speaker adaptation training and discriminative training [Ver04].

2.2.1 Hidden Markov Model

The hidden Markov model (HMM), as an extension of the Markov chain, is a powerful tool for modeling time series signals and plays a dominant role in acoustic modeling. In some problems, the underlying generative state sequence cannot be observed directly, as each state generates output signals according to a certain probability distribution, and thus these sequential output signals are also called observation sequence. To model such doubly stochastic processes, HMM extends the Markov chain by assigning a probability density function or probability mass function to each state.

In order to reduce the computational complexity, two assumptions are made in HMM. The first assumption is that the next state depends only upon the current state, this is also called the Markov assumption. The second assumption, known as the independence assumption, is that the current output (observation) is dependent only upon the current state.

Mathematical Formulation

Given a set of a finite number of states $S = (s_1, s_2, ..., s_N)$, a hidden Markov model can be defined as a triple $\lambda = (A, B, \pi)$:

- The initial state distribution $\pi = (\pi_1, \pi_2, ..., \pi_N)$: π describes the probability distribution of the start state q_1 , namely $\pi_i = P(q_1 = s_i), 1 \leq i \leq N$, subject to the constraint $\sum_{i=1}^{N} \pi_i = 1$.
- The state transition matrix $A = (a_{ij})$: a_{ij} denotes the probability of transition from state i to state j, which means $a_{ij} = P(q_t = j|q_{t-1} = i), i, j = 1, 2, ..., N$, subject to the constraint $\sum_{j=1}^{N} a_{ij} = 1$ for all $1 \leq i \leq N$.
- The observation (emission) probability distribution $B = \{b_j(k)\}$: $b_j(k) = P(v_k|q_t = s_j), 1 \leq j \leq N, 1 \leq k \leq M$, the probability of observing the event k given the state j at a certain point. The observation probability distribution can also be continuous, in this case $b_j(k)$ are modeled by the probability density functions.

The above mathematical formulation of the HMM presents all the parameters of a HMM, these can be learned from data through a training process, or alternatively, determined according to a priori knowledge.

2.2.2 Gaussian Mixture Model

The term Gaussian mixture model refers to a probability distribution which is expressed as the weighted sum of K single Gaussian distributions, where each single Gaussian distribution is called a "component" and K is the number of components. By increasing the number of Gaussian components, GMMs are able to approximate any continuous probability distribution arbitrarily closely. The probability density function of a GMM can be represented by the following:

$$\begin{aligned} p(x) &= \sum_{k=1}^{K} p(k)\, p(x|k) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \end{aligned}$$

where μ_k and Σ_k are the mean vector and covariance matrix of the k-th single Gaussian component, π_k are mixture weights, satisfy the constraints $0 \leq \pi_m \leq 1$ for $1 \leq m \leq K$

and $\sum_{k=1}^{K} \pi_k = 1$, π_k can be interpreted as the a priori probability of the k-th Gaussian component.

Parameter Estimation

Given N data points sampled from a GMM, then the parameters of the GMM can be estimated using the Maximum-Likelihood method, with the aim to maximize either the likelihood function $\prod_{i=1}^{N} p(x_i)$ or the log-likelihood function $\sum_{i=1}^{N} \log p(x_i)$. Since it is very difficult to obtain a closed-form solution for this optimization problem, the parameters are usually estimated using Expectation Maximization (EM) algorithm in an iterative fashion [B$^+$06]. The basic principle of the EM algorithm is to start from randomly initialized values for the model parameters Θ and then in each iteration estimate a new set of parameter values Θ' based on the current set of model parameters Θ, which ensures:

$$\sum_{i=1}^{N} \log p\left(x_i|\Theta'\right) > \sum_{i=1}^{N} \log p\left(x_i|\Theta\right).$$

Each iteration of EM training consists of the following two steps:

- E-Step (Expectation) estimates the probabilities that the data point i generated by the component k:

$$\gamma(i,k) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=k}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

- M-Step (Maximization) re-estimates the model parameters using Maximum-Likelihood method with the help of $\gamma(i,k)$ in estimated in E-step:

$$\begin{aligned} N_k &= \sum_{i=1}^{N} \gamma(i,k) \\ \mu_k &= \frac{1}{N_k} \sum_{i=1}^{N} \gamma(i,k)\, x_i \\ \Sigma_k &= \frac{1}{N_k} \sum_{i=1}^{N} \gamma(i,k)\, (x_i - \mu_k)\, (x_i - \mu_k)^T \\ \pi_k &= \frac{N_k}{N} \end{aligned}$$

This process continues until the values of $\sum_{i=1}^{N} \log p(x_i|\Theta)$ converge. However the EM algorithm cannot be guaranteed to converge on a globally optimal solution, since finding a global optimum is basically a NP-hard problem.

In general the EM algorithm is considered to be a strategy for the computing maximum likelihood estimates of statistical models which have latent variables, and besides GMMs the EM algorithm can be applied in many other specific fields [DLR77]. For example the Baum Welch algorithm, based on the EM algorithm, is used to estimate the parameters of HMMs.

2.2.3 GMM-HMM Acoustic Modeling

Due to the temporal order of speech, acoustic modeling most commonly makes use of left-to-right HMMs to model phonemes. A left-to-right HMM allows only the transitions

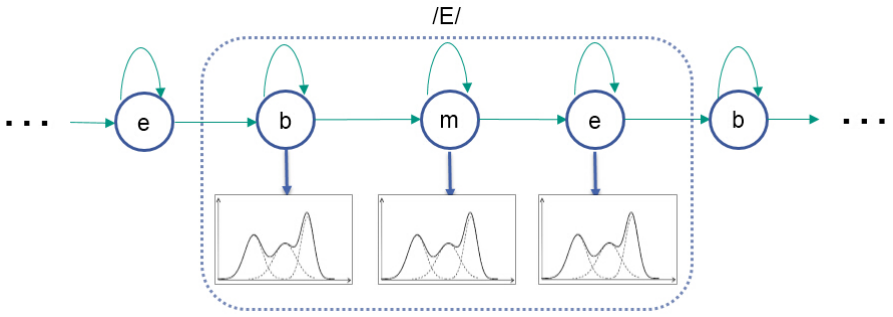

Figure 2.2: *A three-state left-to-right HMM for the phoneme /E/.*

from the current state to itself or to the adjacent state on its right, as shown in figure 2.2. Furthermore these phoneme HMMs typically have three states (beginning, middle and end), where each state models a stage of uttering a phoneme, and the number of HMM states is determined by a priori knowledge. In conventional GMM-HMM systems, the emission probability distributions of states are approximated by GMMs with a diagonal covariance structure to take advantage of their efficient computation. A phoneme HMM can generate a sequence of observation vectors by starting from the initial state, each time generating an observation vector (an n-dimensional feature vector) according to the emission probability distribution of the current HMM state, then moving from the current state to the next in accordance with the state transition matrix.

The phoneme HMMs can be combined to form word-level HMMs, and similarly word-level HMMs can be combined together to form sentence-level HMMs. At this point decoding and alignment algorithms can be applied to either the word-level or sentence-level HMMs. Given a feature vector sequence, decoding algorithms attempt to find the optimal HMM state sequences that generate this feature vector sequence. Alignment algorithms are provided with more information than decoding algorithms, namely the sentence-level HMMs that generate the feature vector sequence, and thus the search space is far smaller than the search space for decoding. Viterbi algorithm can be used for both decoding and alignment problems.

2.2.3.1 HMM-GMM Training

For an observation sequence and a phoneme HMM training algorithm aims to maximize the probability that the phoneme HMM generates the corresponding observation sequence through adjusting the parameters of the HMM. The HMM training problem is normally approached with iterative algorithms, such as the Baum-Welch algorithm which is a variant of the EM algorithm, since there is no analytical solution.

Before training, sufficient training data is needed for the estimation of HMM parameters. The original training corpus contains recorded audio data and the corresponding transcripts. In order to collect training data for each HMM, each frame of feature vectors must be first assigned to an HMM state, this is also known as frame-to-state alignment. The alignment problem can be solved by building a sentence-level HMM for each sentence, then finding the best underlying explanation of the feature vector sequence, namely a most likely states sequence with the Viterbi algorithm. However this approach requires an existing well-trained GMM-HMM system, and thus to bootstrap the training process speech data is first equally aligned. This initial alignment can then be used as the start point for GMM-HMM training. With frame-to-state alignments each phoneme HMM will have the training feature vectors for Baum-Welch training. After several iterations of the Baum-Welch algorithm, the trained GMM-HMM models are used to realign the training data,

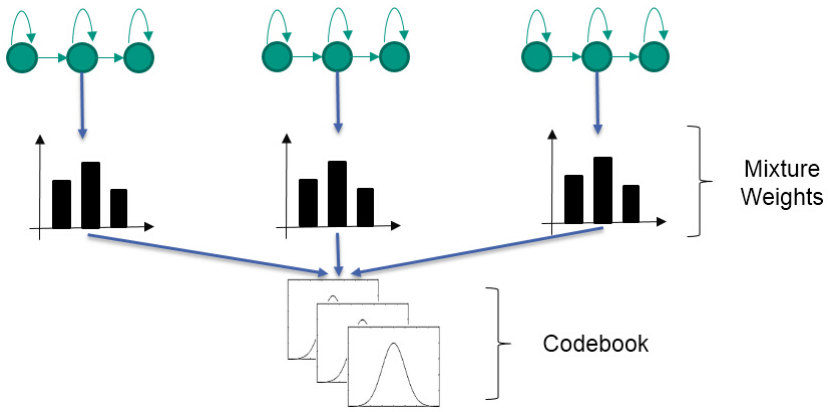

Figure 2.3: *Semi-continuous HMM states share the same codebook, their emission distributions can be represented by the mixture weights.*

generally these realignments are more accurate and can thus be used for the next round of the Baum-Welch algorithm. The training process is repeated until the GMM-HMM system reaches the desired performance.

2.2.4 Semi-continuous HMMs

HMMs employed in ASR can be divided into three categories, namely discrete, semi-continuous and continuous HMMs. Discrete HMMs have discrete output spaces, which means each HMM state has a discrete emission probability distribution. However the acoustic feature vector space is continuous, and thus it is necessary to find a set of representative feature vectors so that every feature vector in the continuous feature space can be represented by one of the them with minimum information loss. Each representative feature vector is also called a codeword, whilst a set of codewords forms a codebook. The feature vector space is partitioned into a number of Voronoi regions, with each represented by a codeword, this step is also called vector quantization.

In continuous HMMs the emission distributions of states are modeled with GMMs as mentioned in section 2.2.3. Continuous HMMs model the output distribution directly, which leads to less information distortion when compared with discrete HMMs and normally results in a better performance of the final ASR system. However continuous HMMs have a large number of free parameters of GMMs (mean vectors, covariance matrices, mixture weights), which require a great amount of training data and considerable computation power for the estimation of the parameters.

As a compromise between the discrete HMM and the continuous HMM, the semi-continuous HMM was first proposed by [HJ89]. The idea behind this model is to replace feature vectors in the codebook of discrete HMMs with Gaussian PDFs, with each single Gaussian analogous to a codeword in discrete HMMs. The GMMs assigned to HMM states as emission distributions share the same codebook, and thus different GMMs are distinguished by their mixture weights. In this way the number of parameters decreases dramatically in comparison to continuous HMMs, yet the quantization error suffered by discrete HMMs is also reduced. Moreover, by increasing the size of the codebook semi-continuous HMMs have equivalent modeling power to continuous HMMs.

2.3 Context Dependent Acoustic Model

Coarticulation refers to the effect where the speech sound of a phoneme is influenced by, and becomes more like, its neighbouring phonemes [CB10]. From the perspective of vocal

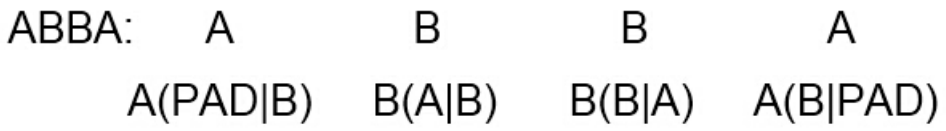

Figure 2.4: *The word ABBA modeled with four triphones.*

mechanism coarticulation is due to the fact that vocal organs can change only continuously during utterance of speech sounds. In continuous speech the coarticulation effect is very common, however the GMM-HMM models discussed in section 2.2.3 cannot handle the coarticulation effect directly, because coarticulation is contrary to the independence assumption which assumes that the outputs (observations) of HMMs are independent of each other. Thus it is necessary to build a context dependent model to take coarticulation into consideration. The state-of-art approach to deal with coarticulation is using GMM-HMM to model polyphones instead of phonemes (monophones), a polyphone is a phoneme in a context with a certain length. Three most common polyphone models are triphone models, which take one phoneme in the left and right context into consideration, with quinphone and biphone models performing very similarly.

As an illustration, the word ABBA can be modeled with four monophones or four triphones as shown in the figure 2.4, where PAD denotes a filler phoneme for unknown contexts and the triphone B(A|B) means the phoneme /B/ with left phoneme context /A/ and right phoneme context /B/. Furthermore triphone B(A|B) may sound different to triphone B(B|A) because of coarticulation. Therefore phonemes in different contexts are modeled independently with different HMMs, thus such systems are also called context dependent GMM-HMMs (CD-GMM-HMM). In order to train polyphone HMMs, the training data for each polyphone should first be collected, before each polyphone HMM is trained on its data. The training algorithms for monophone and polyphone HMMs have no difference in principal, however CD-GMM-HMM systems suffer the problem that the training data for most polyphones is not sufficient for a robust training.

2.3.1 Decision Tree based State Tying

Polyphone models can greatly improve the accuracy of acoustic modeling, however this also dramatically increases the complexity of the system. In the English language there are about $50^3 = 125000$ triphones, however a great number of them have insufficient training data or never even appear in the training corpus. An effective approach to solving this problem and reducing the model size is parameter sharing (parameter tying), where similar acoustic units share their training data and have identical parameters.

Parameter sharing can happen at three levels:

1. Tying Gaussians (semi-continuous HMMs): GMMs employed to model emission distributions of different HMM states share a set of single Gaussian components (codebook), different GMMs have different mixture weights.
2. Tying states (clustered state): Different HMM states are tied to an identical emission distribution, these tied (clustered) states are the so-called senones.
3. Tying HMMs (generalized triphones): similar polyphones are modeled with the same HMM.

State tying has the advantage of flexibility and fine-granularity compared with other parameter sharing strategies. Figure 2.5 shows an illustration of state tying: The beginning states of triphones E(P|M), E(L|K) and E(O|M) share the same GMM whereas the middle states of E(P|M) and E(L|K) are tied to another GMM.

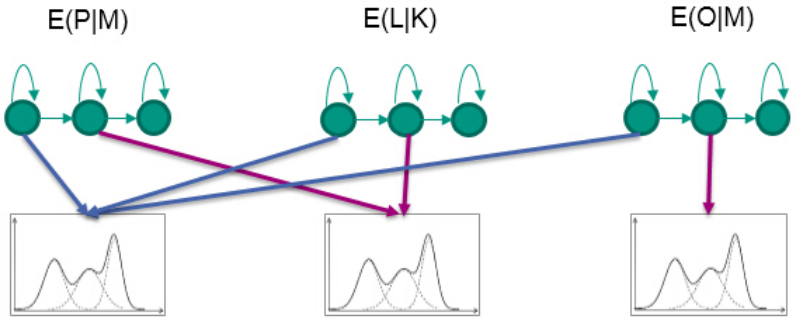

Figure 2.5: *State tying: different HMM states can be tied to the same GMM.*

2.3.1.1 Phonetic Decision Tree

In most state-of-the-art ASR systems, as well as in this work, decision tree based state-tying [YOW94] is employed as the standard approach for context dependent acoustic modeling. A phonetic decision tree, also called a cluster tree, is a binary tree used to perform polyphone HMM state clustering. Every inner node of a phonetic decision tree is bound with a "Yes / No" question. Normally for each state of a phoneme HMM (three states: beginning, middle, end), a decision tree is built to cluster the derived polyphones states that appear in the training data. The leaves of a decision tree are the resultant polyphone state clusters, all the polyphones states in the same leaves can be considered to be similar, thus they can be modeled with the same GMM and share their training data, the leaves of a cluster tree are also regarded as senones.

Whenever the emission distribution of a polyphone state is required during the decoding or alignment procedure:

1. Find the corresponding decision tree of its center phone state, start from the root of the tree.
2. At each inner node, if the answer to the attached question is no, move to left child node, otherwise right child node.
3. Continue until a leaf is reached, then the emission distribution of the associated senone can be used by the given polyphone state.

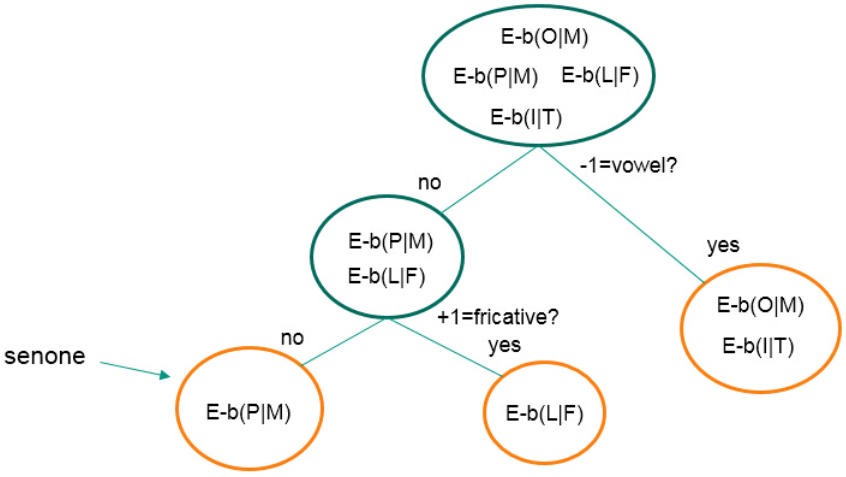

Figure 2.6: *An example of the cluster tree for the triphone states derived from the monophone state E-b.*

2.3.1.2 Decision Tree Building

The cluster tree building process proposed by Bohl et al. in their work [BdG+91] is data-driven and works top-down. Before clustering, the training data must be collected for each appearing polyphone state. At first, the root of a decision tree contains all the triphone states collected from the training data, then a tree can grow by splitting one of its leaves according to a question from the predefined question pool. Questions in the question pool are about the context information, and can be answered only with yes or no, e.g. "Is the left context a vowel" (-1=vowel?). A leaf of a decision tree can be split into two child nodes by putting its contained states with the answer "no" in the left child node, and states with the answer "yes" in the right.

Each time a leaf and a question are selected to perform the splitting, so that the splitting can result in the two most distinct child nodes of all possible splitting, with respect to a certain distance measure, for example entropy distance, Euclidian distance, Kullback-Leibler divergence [Run07], etc. In this way a decision tree grows until a certain stopping criterion is satisfied, then the clustering process stops. The stopping criteria could be when: *a*) after splitting the number of training samples in either child node is less than a certain number; *b*) the information gaining for the splitting is below a threshold; or *c*) the desired number of tree leaves is reached.

2.3.2 Entropy Distance

As briefly discussed in section 2.3.1.2, when decision trees are used for clustering states, a distance measure is needed, so that each splitting can result in two most distinct child clusters. The aim of the distance measure is to quantify the similarity or dissimilarity between two clusters of polyphones. A number of distance measures have been proposed, and usually with different distance measures, the clustering process will lead to different decision trees, which will also influence the performance of the final ASR system [Stu09]. In this work, entropy distance is employed as the distance metric for clustering, and is discussed in more detail in the next section.

2.3.2.1 Entropy of a Discrete Distribution

Given a discrete probability distribution P of n events, where p_i is the probability of the i-th event, the term entropy [Sha01] can be used to measure the uncertainty, or information content, of this distribution. It is considered that random events with lower probability are more uncertain and contain more information. Thus the self-information of a random event is defined as:

$$I(p_i) = -\log(p_i).$$

Self-information is a measurement of uncertainty (information content) of a random event. For example, a certain event, namely an event with the probability 1, has the lowest uncertainty 0, and the uncertainty increases as the probability of the event decreases.

Furthermore the entropy of a discrete probability P is defined as the expectation of self-information of random events, thus entropy can also be understood as the average uncertainty of this distribution:

$$H(X) = \sum_{i=1}^{N} p_i I(p_i) = -\sum_{i=1}^{N} p_i \log(p_i).$$

It can be proved that the uniform distribution has the maximum entropy, which is also consistent with intuition.

2.3.2.2 Information Gain

In 1986 Quinlan proposed a general decision tree building algorithm called the Iterative Dichotomiser 3 (ID3) [Qui86]. When splitting the decision tree, the ID3 algorithm measures the distance between two tree nodes based on entropy, and thus the distance measure employed in ID3 is also called the entropy distance, this is discussed in detail in the following section.

If a discrete probability distribution P can be split into a set of sub-distributions $T = \{P_1, P_2, ..., P_M\}$ according to a certain attribute or with the help of some information, namely:

$$P(x) = \sum_{i=1}^{M} \pi_i P_i(x),$$

where π_i can be interpreted as the a priori probabilities of sub-distributions.

Then the Information Gain can be defined as the loss of entropy after splitting the distribution P:

$$IG(A, P) = H(P) - \sum_{i=1}^{M} \pi_i H(P_i),$$

where A is the attribute that splits P and $\sum_{i=1}^{M} \pi_i H(P_i)$ can be regarded as the average entropy of the sub-distributions. Therefore information gain can be interpreted as the uncertainty loss (entropy loss) when we possess knowledge of the a priori probability of each sub-distribution and the probability that a given instance is generated by each sub-distribution.

2.3.2.3 Information Gain as a Distance Measure

Given two discrete probability distributions P and Q, the distribution of the union of P and Q is estimated as:

$$(P + Q)(i) = \frac{1}{n_P + n_Q}(n_P P(i) + n_Q Q(i)),$$

where n_p and n_q are the numbers of training data points sampled from distributions P and Q respectively, and $n_p/(n_p + n_q)$ is the estimation of the a priori probability of the class P. It can also be considered as that $P + Q$ is split into two sub-distributions P and Q.

Entropy distance can be used to measure the discrepancy between P and Q as:

$$d(P, Q) = (n_P + n_Q)H(P + Q) - n_P H(P) - n_Q H(Q).$$

The formula above is used to calculate the so-called weighted entropy distance, since the actual value of $d(P, Q)$ is the information gain from splitting $(P + Q)$ into P and Q multiplied by the total number of data points sampled from distributions P and Q. This will ensure that models that have more training data have greater opportunity to be split since they are more reliably estimated.

2.4 Word Error Rate

After the acoustic model has been successfully trained it is integrated with the language model, the decoder and other components of an ASR system in order to perform speech recognition tasks. The performance of different acoustic models can be compared by measuring the recognition accuracy of the integrated ASR system with other parts of the system remaining identical.

Word error rate (WER) is the most common metric for recognition performance, and is also used in this work to compare the performance of the two approaches for cluster tree building. WER is defined as the minimum edit distance between hypotheses generated by ASR system and reference transcriptions, namely the minimum number of edit steps (substitutions, deletions, insertions) normalized by the length of the reference transcription:

$$WER = \frac{S + D + I}{N},$$

where S, D and I denote the number of substitutions, deletions and insertions respectively, N is the number of words in the reference transcription. WER can be efficiently calculated through the dynamic programming algorithm, where a lower WER generally indicates a better performance of the ASR system.

3. Deep Neural Network

3.1 Artificial Neural Networks

An artificial neural network (ANN) is a computing model motivated by the human brain, like the human brain it is composed of a large number of interconnected computing units called nodes or neurons.

The neuron is the smallest computing unit of an ANN. A neuron can connect with other neurons, and receive signals from them, which are regarded as the input of the neuron. The task of a neuron is to calculate its output from its input. Taking the neuron i for example, x_{ij} denotes the input signal sent by neuron j, the weight w_{ij} represents the connection strength between neuron i and neuron j, thus $x_{ij} * w_{ij}$ is the weighted input signal of neuron i received from neuron j. The total input of neuron i is obtained by summing all its weighted input signal, which is $\sum_j x_{ij} * w_{ij}$. The net input of neuron i net_i depends not only on the total input signal, but also on a threshold value θ, whose inverse is often called the bias b: $net_i = \sum_j x_{ij} * w_{ij} + b$. Finally the activation (output) of neuron i can be expressed as:

$$o_i = \sigma(\sum_j x_{ij} * w_{ij} + b),$$

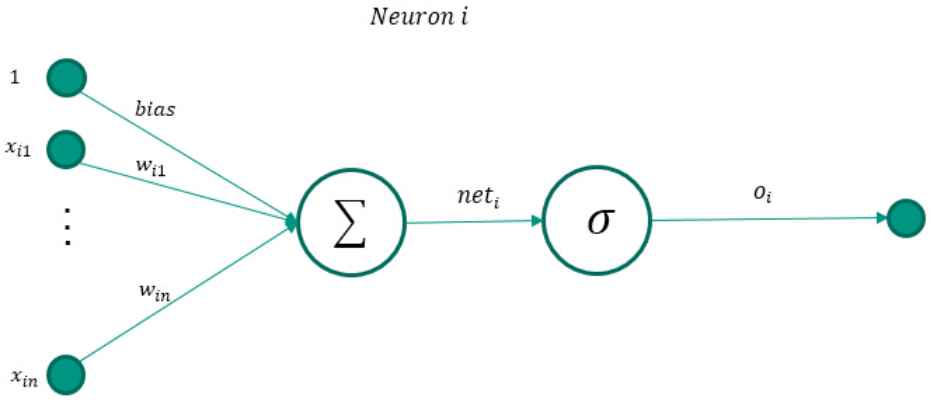

Figure 3.1: *Structure of an artificial neuron, the bias of a neuron can also be regarded as the weight of an input with a constant value of one.*

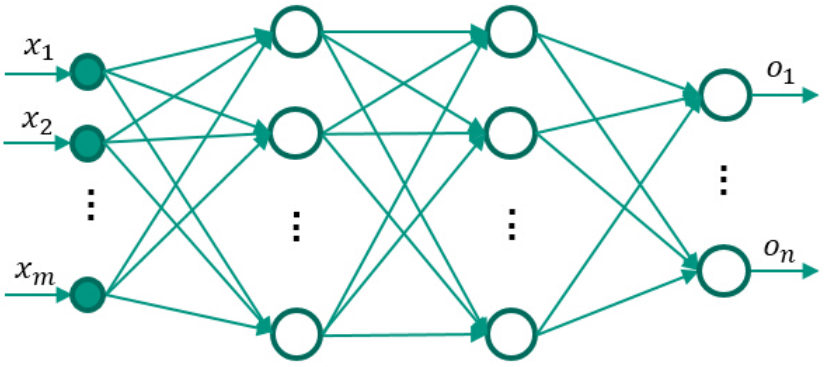

Figure 3.2: *Architecture of an MLP with m input neurons, two hidden layers and n output neurons.*

or $o_i = \sigma(net_i)$, where σ is the activation function and the bias value b allows the activation function to shift along the x-axis. The most commonly used activation functions are sigmoid, tanh, rectified linear unit (ReLU), etc., activation functions are in a position to provide nonlinearity to ANNs. Recent studies [MHN13] [ZRM+13] have shown the superiority of ReLU activation functions over traditional sigmoid functions with respect to training time and generalization of neural networks which leads to a reduction of WER of the final ASR system.

Although the function of a single neuron is quite simple, a neural network composed of a large number of neurons has the ability to approximate highly nonlinear mapping and perform complex tasks such as classification, prediction and clustering. There are different types of ANN models, and one of the most widely used ANN models is the feedforward neural network, which often known as the multilayer perceptron (MLP). An illustration of an MLP is shown in figure 3.2. However networks with feedback are also possible, these are called recurrent neural networks, neurons in a RNN may connected to form directed cycles.

As the name implies the neurons of an MLP are arranged in multiple layers, the MLP has three types of layers - the input layer, the hidden layer and the output layer. Each layer is fully connected with its neighbouring layers. Signals are propagated in one direction from the input layer through one or more hidden layers to the output layer. With the exception of the neurons in the input layer, each neuron in an MLP can have a nonlinear activation function, and thus the input layer is often not included in the number of layers.

3.2 ANN Training

Learning ability is one of the most important and remarkable features of neural networks. In an ANN the weights and bias values of the neurons are initialized with random numbers, afterwards ANNs adjust weights and bias values through learning or training procedure, so that ANNs have the ability to perform complex tasks. ANNs' learning procedure requires a large amount of training data, it is necessary for each training sample to have a label, which is the target of the ANN output, for supervised learning, or else it is unsupervised learning. In the case of supervised learning the ANN produces an output for each training sample in the training dataset, and error can be calculated by comparing this output to the expected target, after which the parameters of the ANN can be adjusted to reduce the error.

Given the network target $\mathbf{t}$ for the input vector $\mathbf{x}$ and the actual output $\mathbf{o}$, error E for

input $\mathbf{x}$ can be measured via various metrics. One of the most common error functions is the sum-of-squares error function:

$$E_x(\mathbf{w}) = \frac{1}{2} \sum_{k \in outputs} (t_{kx} - o_{kx})^2,$$

where $\mathbf{w}$ denotes the parameters of the ANN, and the total error on training database X is:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k \in X} \sum_{k \in outputs} (t_{kx} - o_{kx})^2.$$

However the sum-of-squares error function might not be the best choice for classification problems, the cross entropy error function is normally used for ease of computing:

$$E_x(\mathbf{w}) = - \sum_{k \in outputs} \left[t_{kx} \log(o_{kx}) + (1 - t_{kx}) \log\left(1 - o_{kx}\right) \right].$$

In classification problems the outputs of the neural network are the posterior probabilities of the classification classes given an input vector, each output neuron corresponds to a specific classification class. Thus the output value of each neuron should be between zero and one, and the sum of all output values should be one. To ensure this, the output layer normally uses softmax functions as activation functions:

$$\sigma(net_i) = \frac{e^{net_i}}{\sum_o e^{net_o}}.$$

For a given input vector with a class label i, since each output neuron corresponds to a class, then the target output of i-th output neuron is one and the target outputs of all other output neurons are zero.

The training of an ANN is principally an optimization problem, namely searching for a set of parameters (weights and biases) in the parameter space so that the error $E(\mathbf{W})$ is minimized. However the objective function is in most cases non-convex, and for such optimization problems it is impractical to find an analytical (closed-form) solution, thus the gradient-descent method is often used to acquire an approximately optimal solution. The error backpropagation (BP) algorithm, proposed by Rumelhant and Hinton, is one of the most widely used and successful gradient-descent based algorithms for MLP training [RHW88]. The BP algorithm is an iterative algorithm, in each iteration the parameters of the ANN are updated to get closer to a local minimum, the learning process continues until the error reduction between two iterations decreases to a threshold.

A variant of the BP algorithm is Mini-Batch Gradient Descent (MGD) [Mit97], instead of using all training samples for each ANN weight update to minimize the total error $E(\mathbf{W})$, MGD randomly selects a mini-batch of training samples B then updates the weights to minimize $E_B = \sum_{x \in B} E_x$. One advantage of MGD is that it converges faster in the case of a large training database. The first step of MGD is to split the training database into a set of non-intersecting mini-batches of equal size, denoted by $\mathfrak{B}$, a typical mini-batch size is between 2 and 1024. Next, the mini-batches are shuffled in order to reduce the risk that a sequence of incorrectly labeled or corrupted training data sends the searching procedure in a direction away from the local minimum.

Each iteration of the MGD algorithm consists of the following steps:

1. Randomly select a mini-batch $B \in \mathfrak{B}$ and calculate the output of the neural network for all $x \in B$.
2. For each neuron calculate the error terms δ_i, which indicates how much influence a neuron has on the output error, in the case of the cross entropy error function and softmax output layer:
   - for output neurons: $\delta_i = \frac{1}{|B|} \sum_{x \in B} t_{ix} - o_{ix}$
   - for input neurons: $\delta_i = \frac{1}{|B|} \sum_{x \in B} \left[o_i(1 - o_i) \sum_{k \in Downstream(i)} \delta_k w_{ki} \right]$, where Downstream(i) denotes the set of neurons that receive input signal from the neuron i.
3. Compute the desired change in weight $\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$, and update the weights: $w_{ij} \leftarrow w_{ij} + \eta \delta_i x_{ij}$, where η is the learning rate, which determines how much the weights are actually modified by the term Δw_{ij}.
4. Repeat steps 1-3 until all $B \in \mathfrak{B}$ have been used once.

Each iteration of BP training is also called an epoch, and for each epoch every training sample is passed through the neural network, the training of ANN will continue for a number of epochs until E_B converges.

3.3 Deep Learning

ANNs with more than one hidden layer are referred as Deep Neural Networks (DNN). Although [HSW89] has proved the universal approximation capability of a single hidden layer feedforward neural network, such approximations may not be efficient. [BL+07] states that deep architecture can efficiently express some complex functions, however an inappropriate shallow architecture may require exponentially more computing units. These studies indicated the limitations of shallow architectures, and drew more attention to discovering the abilities of deep neural networks. Moreover, deep network allows learning of hierarchical representations of input data, which means DNNs learn the representations of data layer-wise, higher layers learn more abstract features composed of lower layer features and therefore deep learning is also known as representation learning. Evidence that supports representation learning can be that recent research on human and non-human primate visual system have shown that brains process visual information in a hierarchical architecture, starting from edge detection, and progressing to simple and then complex shape detection [SKK+07].

Although the superiority of deep architectures has long been known, and various attempts have been made to train DNN, the training of DNN remained a difficult problem until Hinton introduced a novel approach for Deep Belief Net training in his work [HOT06] in 2006. Since training a DNN is fundamentally a non-convex optimization problem with numerous local minima, the difficulty of DNN training lies in the fact that gradient-descent method such as the BP algorithm can easily get trapped into a local minimum. Deep learning algorithms try to alleviate this problem by first pre-training the DNN in a greedy layer-wise manner with an unsupervised training algorithm, then fine-tuning the weights using the supervised BP algorithm. [EBC+10] states that comparing with traditional methods in which the weights are just initialized randomly, the initial weights obtained from pre-training provide a better starting point for searching in the weight space with the BP algorithm. This is because pre-training can play the role of a regularizer which restricts the search space of the subsequent fine-tuning to a small region of the weight space, enabling DNNs a better generalization performance.

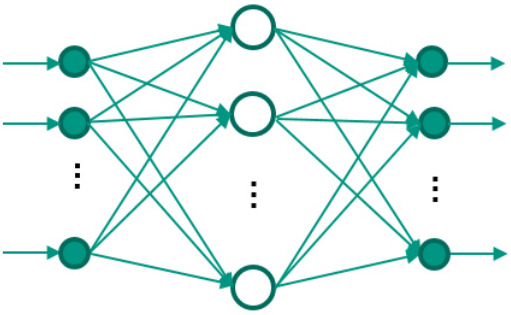

Figure 3.3: *Architecture of an autoencoder, the input and output layers have the same size.*

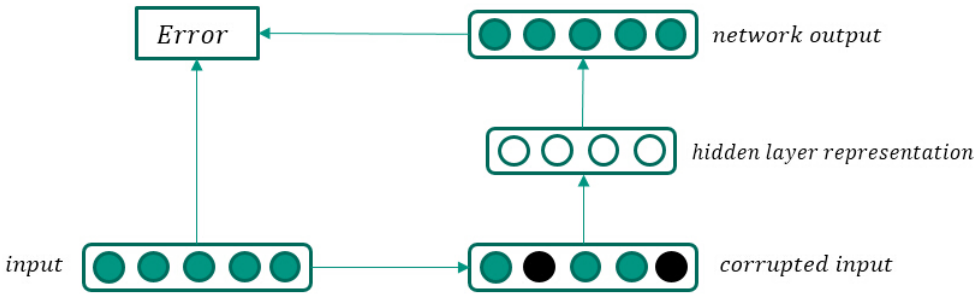

Figure 3.4: *Training a denoising autoencoder, the black color denotes that the input values are set to zero.*

The next section describes a practical approach to pre-training which is used in this work, it is based on the stacked denoising autoencoders introduced by Vincent et al. in their work [VLBM08]. Another successful approach to pre-training is based on the restricted Boltzmann machine (RBM), which was proposed by Ackley, Hinton et al. in [AHS85].

3.3.1 Stacked denoising Autoencoders (SdA)

Deep learning is also known as representation learning, its aim is to use a large amount of data to pre-train the network, so that each hidden layer can learn more robust and abstract representations of input feature vectors. A criterion of robust representation can be that it conserves so much original information that the input vector can be somehow reconstructed by only relying on the features learned in the hidden layers, and the autoencoder was proposed based on this idea [BLP+07]. An autoencoder is a single hidden layer MLP, whose input layer and output layer have exactly the same size.

Autoencoders can be trained with the BP algorithm, the target of each training sample is the training sample itself, in other words an autoencoder learns to approximate an identity function, and the activation of the hidden layer is the coding or representation of the input. If the number of neurons in the hidden layer is greater than the number of neurons in the input layer, and the autoencoder can approximate the identity function very closely, then a compact representation of the input can be learned in the hidden layer, this is equivalent to a dimensionality reduction of the input data. The size of the hidden layer can also be larger than the size of the input layer and in this case most neurons in the hidden layer are normally restricted to be inactive. This variant of the autoencoder is called the sparse autoencoder.

Another variant of the autoencoder is the denoising autoencoder (dA) [VLBM08], a denoising autoencoder extends the conventional autoencoder with a denoising criterion. When training a denoising autoencoder, for a given input vector some of its values are randomly

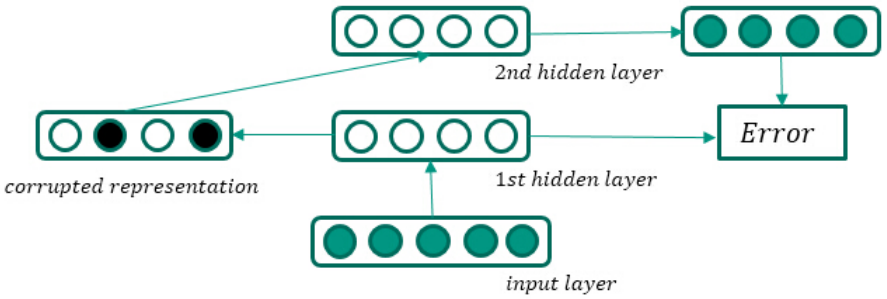

Figure 3.5: *Training a stacked denoising autoencoder.*

set to zero, after which this corrupted input vector is placed on the input layer with the target of the output layer defined as the original input vector, since the task of a denoising autoencoder is to reconstruct the original input. It is shown that the representations learned by denoising autoencoders are more robust, some mathematical proofs can be found in Vincent's work [VLBM08].

The hidden layers of DNN can be initialized with SdAs, the training of SdAs is the so called pre-training of DNNs. The training algorithm is as follows:

1. Train the first hidden layer of an SdA as a denoising autoencoder on all the training data.
2. Train the n-th hidden layer as a dA with the corrupted output of (n-1)-th layer as the input to reconstruct the uncorrupted output of (n-1)-th layer.
3. Repeat step 2 until all hidden layers are trained.

Once the training of the SdA is complete, a logistic regression layer can be added to the topmost layer, after which the whole neural network processes the input data as an MLP. With labeled training data the weights of the deep neural network can be fine-tuned, which is normally approached via the BP algorithm.

3.4 DNN-HMM Hybrid System

In the last part of this chapter a successful application of DNNs in speech recognition, namely using a combination of DNN and HMM as the acoustic model, is discussed. There are two main ways to employ DNNs in acoustic modeling, one is the tandem system [HES00], while the other is the DNN-HMM hybrid system. In a tandem system the DNN is exploited as a feature extractor, the outputs of the neural network or the activations of a narrow hidden layer (bottleneck features) are used as feature vectors, which are normally concatenated with conventional acoustic features e.g. MFCC, by the GMM-HMM system. In a DNN-HMM hybrid system the DNN replaces GMMs to estimate the emission distributions of HMM states.

Early in the 1990's Bourlard and Morgan introduced the ANN-HMM hybrid system in their work [MB90]. However, due to the limited computational power and the lack of pre-training algorithms the powerful classification ability of deep neural networks were not fully exploited, and thus the ANN-HMM hybrid system had no distinct advantage over the conventional GMM-HMM system for a period of time.

DNN-HMM hybrid systems take advantage of the DNN to estimate the a posteriori probabilities of the monophone HMM states q_i when a feature vector of speech signal x is

observed, namely $P(q_i|x)$. After applying the Bayes' rule we obtain the emission probability:

$$p\left(x|q_i\right) = \frac{P\left(q_i|x\right)p\left(x\right)}{P\left(q_i\right)},$$

where $P(q_i)$ is the a priori probability of the state q_i, which can be estimated from the training data, and $p(x)$ is the probability density function of feature vector x instead of a probability mass function, since the infinite number of feature vectors leads to the probability of each feature vector being zero. The actual value of $p(x)$ is not necessary for decoding or alignment algorithms, thus $p(x)$ is usually set to a constant. Finally the emission distribution of the HMM state q_i can be approximated by:

$$p\left(x|q_i\right) \propto \frac{P\left(q_i|x\right)}{P\left(q_i\right)}.$$

As stated in [YD14], taking not only the current feature but also features in the context (normally 4-6 on the right and left respectively) as inputs for the DNN is helpful, because DNN can thus also take the correlation between the consecutive feature vectors into consideration. Furthermore, Dahl and Yu have shown in [DYDA12] that DNN-HMM hybrid systems can be extended to context dependent models (CD-DNN-HMM) by estimating the a posteriori probabilities of senones (tied polyphone states), and that neural networks with deeper architecture and additional pre-training are more powerful. These two improvements resulted in a significant reduction of recognition error rates, thus the CD-DNN-HMM hybrid system has obvious superiority over the conventional GMM-HMM system, and has become the state-of-the-art approach.

4. DNN based Cluster Tree

The previous chapters give background knowledge which helps to understand the problem this work tries to solve. In this chapter we present two approaches to solving the problem. The conventional solution using GMM based clustering is discussed first, and after that the novel DNN based approach is introduced.

4.1 Problem Analysis

As described in section 2.3.1.2, the clustering process is data-driven, thus distance measures are used to measure the distance between two sets of polyphone states, which are represented by the corresponding feature vectors collected from the training corpus. The feature vector space of a polyphone state can be regarded as all the feature vectors that this state can possibly generate. Since the feature vector spaces of polyphone states are continuous, thus they are normally modeled by probability density functions, which are also called emission distributions, in most cases emission distributions are approximated by GMMs. In continuous HMMs the feature vector spaces of each HMM state are modeled separately, each HMM state has its own set of GMM parameters estimated on their training data. In semi-continuous HMMs, all HMM states share the same codebook, which is a set of single Gaussians, and their Gaussian mixture weights are estimated on the training data, thus the feature vector spaces of HMM states can be represented by the sets of mixture weights.

In section 2.3.2, the entropy distance was discussed. Entropy distance is based on the entropy of discrete probability distributions, and defined as the information gain (entropy loss) by splitting a discrete distribution into a set of sub-distributions according to a specific attribute. However, entropy distance cannot be used directly for building phonetic decision trees (described in section 2.3.1.2) because probability distributions of data are assumed to be discrete by entropy distance. In contrast to this, the acoustic feature vector space of HMM states is continuous.

In this work two approaches to solve this problem are investigated, the one discussed first in the following section is the conventional GMM based approach which makes use of a semi-continuous HMM system, while in the next a novel DNN based approach is introduced.

4.2 Previous Work: GMM based Clustering

In semi-continuous HMMs, although the emission distributions of HMM states are still modeled by GMMs, emission distributions can be represented just by their mixture weights and different emission distributions can be compared by their mixture weights, since all HMM states share the same codebook. Furthermore, the mixture weights of polyphone states with sum one can be regarded as the probabilities of a discrete distribution, with each mixture weight interpreted as the a priori probability of a codeword (Gaussian). Since acoustically similar polyphone states (with the same center phone state), are considered to have similar feature spaces which are modeled by GMMs using the same codebook, thus their mixture weights should also be similar. We have seen the distance of discrete distributions can be measured using entropy distance, as a result, the similarities between underlying GMMs of HMM states can be measured using entropy distance via their mixture weights.

As an improvement to conventional semi-continuous HMMs where all states share the same codebook, in this work, for each phoneme (monophone) HMM state a separate codebook is trained with the k-means algorithm. Hence in our system no Gaussian mixture sharing takes place for context independent acoustic modeling, since each monophone state has its own codebook. Such separation makes sense, since different phoneme states have different feature vector distributions. However when it is extended to the context dependent acoustic model, all polyphone states appearing in the training data are collected first, then those polyphone states derived from the same monophone states share a codebook, and each polyphone state's mixture weights is estimated from its own training data. As we grow a separate cluster tree for the polyphone states with the same center phone state, and the polyphone states attached to a cluster tree share the same codebook, thus the distances between them can be measured by entropy distance while cluster tree building.

4.3 DNN based Clustering

The previous section has shown different polyphone states can be compared through their mixture weights when semi-continuous HMMs are employed for acoustic modeling. However this GMM based approach has some shortcomings, as discussed in [HDY+12], GMM is not believed to be the best model for acoustic feature spaces, and hence the final decision tree based on GMMs may not achieve clustering with the desirable performance. Furthermore, current ASR systems still require a GMM library merely out of the need for cluster tree building, a GMM-free cluster tree building algorithm can simplify ASR systems by removing the GMM library from within. In this section, a novel GMM-free approach is introduced, which leverages the powerful classification ability of DNN.

4.3.1 Context Independent DNN Training

Before performing the state clustering, a discrete probability distribution should be calculated for each polyphone state to characterize their feature vector spaces, in our novel approach this is realized by making use of a context independent DNN. As introduced in section 3.4, a CI-DNN is used to estimate the a posteriori probabilities of monophone states given an input feature vector. Thus a CI-DNN can be trained before cluster tree building, without taking context information into account. Normally a CI-DNN is trained with the BP-algorithm in a supervised fashion, and additional pre-training is usually also helpful and able to give a better performance to the CI-DNN.

To bootstrap the training of a CI-DNN, alignments are necessary to provide the target state labels of the DNN output. In the case of CI-DNN, alignments assign each feature vector to a monophone state. Initial alignments are usually obtained with the help of a

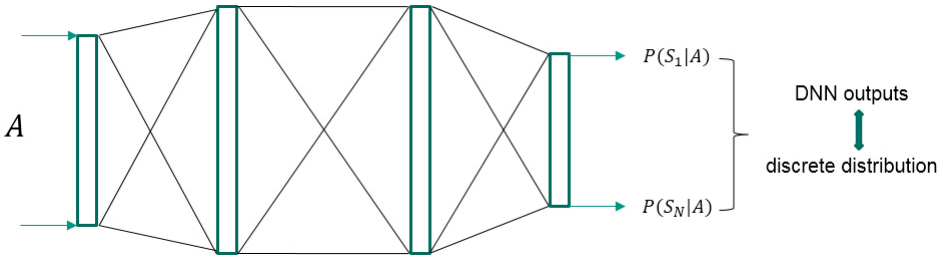

Figure 4.1: *The architecture of a CI-DNN with two hidden layers, where $S_1, ..., S_N$ are monophone states. The input of a CI-DNN is the feature vector A, the output is the a posteriori probabilities of HMM states.*

well-trained GMM-HMM system, nevertheless CI-DNN training can also be flat started without an existing system such as GMM-HMM. [ZW14] has shown CI-DNN training can be bootstrapped with equally aligned training data, after which the classification accuracy of CI-DNNs can be improved by iteratively realigning the data and training new CI-DNNs on realignments. [SHBL14] randomly initialized a CI-DNN with weights near zero, then this untrained CI-DNN was used to align the training data, by doing this initial alignments were obtained which can be used as start point of iterative training of the CI-DNN. CI-DNN-HMMs cannot achieve a performance comparable to CD-DNN-HMMs, and thus CI-DNN-HMMs are normally used to generate initial alignments to bootstrap CD-DNN-HMM training. The following section discusses the possibility of using CI-DNN for decision tree building.

4.3.2 Cluster Tree Building

In a CI-DNN, each output neuron corresponds to a phoneme HMM state, and the output of an output neuron represents the a posteriori probability of the associated HMM state (shown in figure 4.1).

For each input feature vector a discrete probability distribution of underlying generative monophone states is calculated through a well-trained CI-DNN. With the help of existing frame-to-state alignments each polyphone state possesses a set of feature vectors assigned to it, and feature vectors belonging to the same polyphone state should generate almost identical DNN outputs. This is reasonable, because DNN has the ability to layer-wise abstract input features, and each layer of a DNN learns more abstract representations based on the representations delivered by the previous layer, thus DNNs have the ability to extract similarities and deal with variations of different instances from the same class. The average CI-DNN output of a polyphone state can be calculated by summing the CI-DNN output vectors of all the feature vectors assigned to it, then dividing the sum by the count of feature vectors. The CI-DNN outputs of every feature vector should scatter around the average CI-DNN output with little variance, and thus average CI-DNN output can be used to characterize the feature vector space of a polyphone state.

In the average CI-DNN output, the monophone state from which this polyphone state derives always has the greatest a posteriori probability and furthermore, since speech sound of a phoneme is influenced by its neighbouring phonemes through coarticulation, this may also lead to higher a posteriori probabilities for its neighbouring phonemes. Our novel approach rests on the idea that similar polyphone states should also have similar average CI-DNN outputs, moreover the average CI-DNN output can also be regarded as a discrete distribution, therefore entropy distance between polyphone states can be measured based on their average CI-DNN outputs. It is still worth considering that using the normalized activations of a bottleneck layer or a larger hidden layer with 200-500 neurons. However

this may not achieve an equivalent performance to using the CI-DNN outputs, since the output layer learns more from the supervised training and hence is able to capture more information of the feature vector space of polyphone states.

5. Implementation

This chapter documents in detail the pipeline to train DNN based cluster trees so that other researchers can rebuild the systems, and the training steps of the conventional GMM based approach as the baseline system. Furthermore, the test setup for the created cluster trees is also presented in this chapter.

5.1 Janus Recognition Toolkit

In this work all experiments, including the training of acoustic models especially cluster tree building, and the evaluation of the ASR systems, are carried out with the Janus Recognition Toolkit (JRTk) which is developed at Karlsruhe Institute of Technology and Carnegie Mellon University [FGH+97]. JRTk is implemented in C language which ensures the high efficiency of the program's execution. JRTk also provides an object-oriented programming interface in Tcl language, which allows the models written in C to be manipulated at a higher level. In JRTk the IBIS one-pass decoder is also included, since performing decoding tasks with this one-pass decoder enables real-time feature of speech recognition [SMFW01]. For preprocessing, acoustic modeling and decoding tasks JRTk is equipped with a range of state-of-the-art techniques which guarantee that JRTk achieves remarkable recognition performance on various recognition tasks. The training of DNNs in this work was performed using a neural network training setup which is based on the Theano library and has implemented most of the common DNN training algorithms [BLP+12] [BBB+10].

5.2 Training Setup

The following sections present the training setup of cluster trees for the novel approach and baseline, including design decisions and concrete parameters. Both triphone and quinphone models are well supported by JRTk, and since the implementation of the triphone model is quite similar to that of the quinphone model, we take only the quinphone model as an illustration for ease of presentation.

5.2.1 DNN based Cluster Tree

5.2.1.1 CI-DNN Training

As introduced in section 4.3, the first step of our novel approach is to train a CI-DNN. DNNs were trained with the neural network training toolkit which took training data

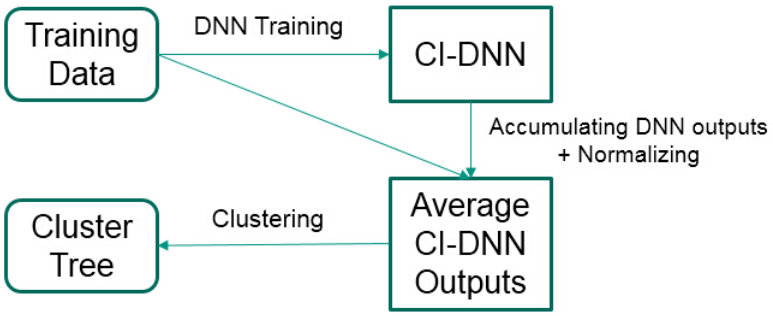

Figure 5.1: *The pipeline to build a DNN based cluster trees. The start point is training data accompanied by frame-to-state alignments. A CI-DNN can be trained based on these alignments, after that we can calculate the average CI-DNN outputs for each quinphone state using the CI-DNN. Finally the clustering algorithm can be performed based on the average CI-DNN outputs, and the end result is the phonetic cluster trees.*

Utterrance Index	Frame Index	Feature Vector	Class Lable
0	0	[0.1, 0.2, ..., 0.1]	2
0	1	[0.3, 0.2, ..., 0.5]	108
1	0	[0.4, 0.8, ..., 0.2]	3
1	1	[0.1, 0.1, ..., 0.2]	23

Table 5.1: *PFile* format and example rows

in the PFile format as inputs. The PFile format[1], developed by ICSI, is used to store acoustic feature vectors and their labels which serve as DNN inputs and the targets of DNN outputs respectively. A PFile file is composed of a number of data rows, table 5.1 shows four example rows from a PFile file. The first entry in each row is the utterance index, followed by the frame index which indicates the position of a frame in an utterance. The next two entries are feature vectors extracted from the speech signal and its class label. In our experiments class labels were monophone states represented by numbers between 0 and 138. As labeling of training samples are necessary for supervised training and determines the classification accuracy of the CI-DNN, we used a well-trained ASR system to align the training data.

Usually the entire training dataset is split into two parts, one part is used for the actual training, while the other is the cross validation set. During the BP training, frame error rates are measured on a cross validation set after each epoch of training, and according to the frame error rates we adjust the learning rate and decide when to stop the training process in order to avoid overfitting.

After creating the PFile training data, a DNN can be trained using the training data. In this work we employed a CI-DNN which had an input layer with 630 neurons, since the input feature vectors were 42-dimensional feature vectors with context ± 7 and four hidden layers, each hidden layer had 1200 neurons, which could provide sufficient modeling ability, and an output layer with 139 neurons, equal to the number of monophone states.

In our experiments each hidden layer was first pre-trained with denoising autoencoder on the training data. Table 5.2 lists some parameters for the SdA pre-training, where the sparsity represents the target mean of activations in the hidden layers and the corruption level denotes the proportion of corrupted input components.

[1] http://www1.icsi.berkeley.edu/Speech/faq/ftrformats.html

Learning Rate	#Mini-batches	Corruption Level	Batch Size	Sparsity	Loss Function
0.01	2000000	0.2	128	0.05	Cross Entropy

Table 5.2: SdA pre-training parameters

Newbob Decay	Newbob Threshold	Batch Size	Activation Function	Loss Function
0.5	[0.005, 0.0001]	256	Sigmoid	Cross Entropy

Table 5.3: BP fine-tuning parameters

After the pre-training of four hidden layers was completed, a logistic regression layer with 139 output neurons was added on top of the hidden layers. Then the DNN was fine-tuned with the supervised BP algorithm. We employed Newbob strategy to adjust the learning rate, the learning rate started with one and when the decrease in cross validation error between the last two epochs dropped below 0.005, the learning rate was multiplied by 0.5, and when the cross validation error decrease was less than 0.0001, the training process stopped.

The BP training lasted for fourteen epochs with the final cross validation error being 0.29, and the ASR system based on the CI-DNN results in a WER of 26.90, such a performance is sufficient to estimate the a posteriori probabilities of underlying monophone states given a feature vector and hence is suitable for cluster tree building.

5.2.1.2 Cluster Tree Building

The next step is to collect the feature vectors for each quinphone state with the help of frame-to-state alignments. For each frame of feature vectors the CI-DNN generates a 139-dimensional output vector. The average CI-DNN output of a quinphone state can be calculated by passing all feature vectors assigned to the quinphone state through the CI-DNN, summing the DNN output vectors and then normalizing this sum by the number of feature vectors.

Concrete implementation consisted of the following steps:

1. First the PFile format training data was created to save the feature vectors in the training corpus and the corresponding labels (quinphone states) based on existing alignments. We then printed the content of the PFile file in sequence using the tools provided by ICSI. The printed PFile file is presented in ASCII symbols, which can be processed by scripts in other programming languages like Python. With the printed PFile we can determine the label (quinphone states) of each feature frame.

2. For each feature frame a 139-dim CI-DNN output vector was computed by passing all feature frames in the PFile file through the CI-DNN in sequence and then saved the CI-DNN outputs to another file. Here we modified the source code of the neural network training toolkit to obtain all output values of the DNN instead of merely the index of the output neuron with the highest output value.

3. The last step was to compute average CI-DNN outputs. Since the PFile file was printed in sequence and the DNN output vectors are also saved in the same order, we were able to find out all the CI-DNN outputs belonging to a certain quinphone state. We then wrote a script to accumulate the CI-DNN outputs for all quinphone states and normalized the accumulated CI-DNN outputs so that the sum of output values equaled one.

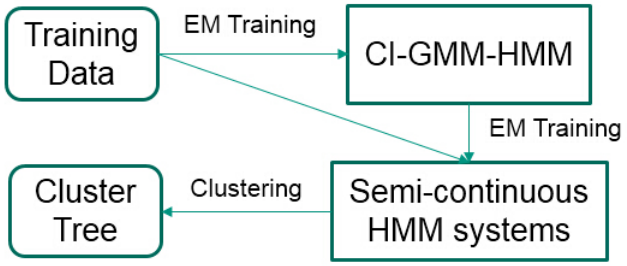

Figure 5.2: *The pipeline for building a conventional GMM based cluster tree. The first step is to train a CI-GMM-HMM system, and then the context independent system is extended to a context dependent semi-continuous HMM system, finally the mixture weights of the semi-continuous HMMs are used to build the cluster trees.*

After performing the above steps, we saved the average CI-DNN outputs and counts for all quinphone states in a file which was later used during cluster tree building.

Besides questions about a category of phonemes described in 2.3.1.2, in our experiments JRTk also asked questions about a specific phoneme and word boundary, for example the question {+2=EH +2=WB} asks whether the second phoneme in the quinphone's right context is the phoneme /EH/ and also at the word boundary. JRTk provides a range of distance metrics like entropy distance and KL-distance to measure the similarity between two sets of polyphone states, in this work we utilized entropy distance for cluster tree building. At the beginning of the clustering process the cluster tree was a root containing all the quinphone states, then the tree was split regarding optimal questions so that each splitting resulted in the two most distinct child nodes, entropy distances of different sets of quinphone states were measured based on the average CI-DNN outputs. The splitting procedure continued until the desired number of leaves was reached and each leaf represented a senone. Previous experiments have shown that the number of senones has a significant influence on the performance of context dependent acoustic models, thus this work investigated cluster trees of different sizes in order to find the optimum number of context dependent states.

5.2.2 Baseline System

Baseline systems implement the GMM based clustering approach described in the section on previous work, some steps of the implementation of the baseline are no different to the DNN based approach, and thus these parts are only briefly mentioned here.

5.2.2.1 Semi-continuous HMM Training

Firstly, a context independent GMM-HMM system was trained with the Baum-Welch algorithm on 42-dimensional LDA features, which means that for each monophone we trained an HMM in which the emission distribution of each state was modeled by a GMM. There were in total 46 German phonemes, including non-speech phonemes like breath, noise and laughter modeled by three-state HMMs, silence was modeled by a single state HMM as well. To ensure sufficiently accurate modeling of emission distributions, each GMM employed 32 Gaussian mixture components. After several iterations of the Baum-Welch training we got the final training results, which were saved in two files - a codebook file containing 139 codebooks (mean vector and covariance matrix) and a distribution file containing the mixture weights corresponding to the 139 codebooks.

In the next step the CI-GMM-HMM system was extended to a context dependent system. The transcriptions were scanned to gather all occurring quinphone states, and then their

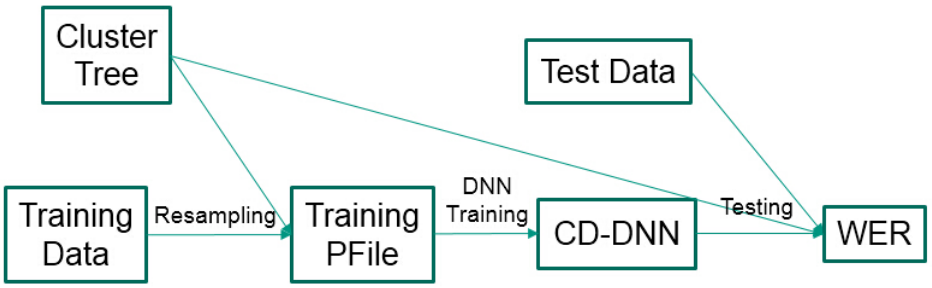

Figure 5.3: *The test setup for the cluster trees. With cluster trees the polyphone states in the alignments of training data are replaced with the leaves in cluster trees (senones). A CD-DNN can then be trained based on these alignments. The trained CD-DNN is integrated into an ASR system as the acoustic model, the final ASR system is tested on test data, and the results can be represented by WERs. Generally by comparing WERs, we compare the performance of different acoustic models.*

feature vectors were collected through alignments. After this we trained tied mixture HMMs for these quinphone states, quinphone states derived from the same monophone states shared the codebook belonging to their center phone state, only the mixture weights of GMMs were trained. Therefore different quinphone states can be compared via their mixture weights.

5.2.2.2 Cluster Tree Building

Entropy distance allows comparison between GMMs through their mixture weights only when they are defined over the same codebook. Our semi-continuous HMM systems fulfilled this constraint since all quinphone states derived from the same monophone state shared the codebook. During cluster tree building mixture weights were used for entropy distance measuring, of note is that in DNN based cluster trees entropy distance was based on 139 dimensional vectors (average CI-DNN output), and baselines utilized 32 dimensional vectors, thus two sets of quinphone states can have two different distances depending on the approach, which can lead to different cluster trees. Cluster trees can directly influence the performance of CD acoustic models, since cluster trees determine which polyphone states are modeled together by a senone and share their training data. In experiments on building CD-DNN-HMM systems, with different decision trees the training data were differently labeled, which resulted in CD-DNNs with different weights and bias values after supervised fine-tuning.

5.3 Test Setup

This section presents the test setup for the generated cluster trees, especially the training of CD-DNN, as it is also part of the testing. Figure 5.3 shows the block diagram of the testing pipeline.

5.3.0.3 CD-DNN-HMM Training

After creating the cluster trees, CD acoustic models can be built based on them, to achieve a better performance we employed the state-of-the-art CD-DNN-HMM as the CD model. The first step of CD-DNN training was also to create the PFile files, however unlike the PFile files for CI-DNN training, the class labels for CD-DNN training are senones rather than monophone states. We used the same training dataset and alignments for the training of CD-DNN as for CI-DNN training.

The inputs of the CD-DNN were 42-dimensional feature vectors with context ±7, and thus its input layer contained 630 neurons, which was the same as CI-DNN. CD-DNN had four hidden layers with 1600 neurons, which were larger than the hidden layers in the CI-DNN. Larger hidden layers can provide more modeling capability - a necessity since the CD-DNN had a larger output layer. The size of the output layer was dependent upon the number of senones, with each output neuron representing a senone. The CD-DNN was pre-trained with SdA and then fine-tuned with the BP algorithm. The training parameters including learning rate, batch size, etc., were identical as for CI-DNN training. After approximately fifteen epochs the BP training was finished, and normally the entire training process for a CD-DNN could take several days as a result of the large architecture of the DNN and the large amount of training data used. A well-trained CD-DNN can be used for emission distribution modeling of senones and combined with HMMs to form a CD-DNN-HMM hybrid system. Cluster trees are also essential when performing decoding and alignment tasks with CD-DNN-HMM systems, as they determine which senones the occurring polyphone states are tied to.

6. Experiments

6.1 Experiment Setup

We have implemented eight DNN based cluster trees in order to test the performance of cluster trees under different configurations, which are quinphone models with 3k, 6k, 9k, 12k, 15k, 18k and 21k tied states respectively and also a triphone model with 3k tied states as an initial experiment. Since our aim is to make a comprehensive comparison between baseline and DNN based approaches under different configurations, we have also built GMM based cluster trees with the same configurations as the DNN based approach.

6.1.1 Training Database and Test Database

The training database, including audio files and the corresponding transcriptions, came from the Quaero training data, broadcast news and audio from the Baden-Wuerttemberg State Parliament [KHM+14]. The training dataset contained about 35 thousand utterances from different speakers which totaled approximately 250 hours. The entire training dataset was split into two parts by extracting the 4th utterance of each speaker, one part, containing 35209 sentences (61562534 frames), was used for the actual training, while the other part was the cross validation set containing 2168 sentences (3747762 frames). From the training dataset we have collected about 65 million frames for 480 thousand quinphone states in total.

For the evaluation of the final ASR systems we used the test set dev2012 for German obtained from the IWSLT Evaluation Campaign 2014, this test dataset contains approximately 120 minutes of audio recordings from six speakers.

6.1.2 Language Model

Language models are utilized in the decoding procedure of the final system to provide the a priori probabilities of word sequences. In our experiments we employed the 4-gram model as our language model [KN95], this 4-gram model was trained on 1.7 billion German words after cleaning and compound splitting, and supplemented with further 118m Google n-grams.

6.2 Initial Tests

It is generally believed that quinphone models can achieve better recognition accuracy than triphone models due to their finer-grained acoustic modeling, which is a fact also

System \ #Senones	3k	6k	9k	12k	15k	18k	21k
DNN based Tree	19.5	19.2	18.9	18.6	18.4	18.3	18.4
Baseline	19.4	19.1	18.8	18.7	18.5	18.4	18.6

Table 6.1: The test results (word error rate) of integrated ASR systems.

proved by this work. We first performed experiments on triphone models with the aim of generating a functioning ASR system using the DNN based cluster tree and testing the system on the test database as fast as possible. In the initial experiments on triphone models we built two cluster trees with 3k tied states for both the DNN based approach and GMM-based approach. This is because the cluster tree building process for triphone models is faster than that for quinphone models, since the number of triphones is much smaller than quinphones. In the experiments we collected about 160 thousand triphone states and about 400 thousand quinphone states from training data. Furthermore, since we employed CD-DNN-HMMs as the CD acoustic model, a DNN with 3k output neurons could be trained within two days, in contrast the training of a larger DNN, say with 18k output neurons, could take about 4 days. The WERs of both triphone based systems were 19.7%, however the quinphone models with the same number of senones had lower WERs, which were 19.5% for DNN based trees and 19.4% for GMM based trees.

6.3 Evaluation

This section presents and explains the test results of the implemented quinphone systems, all trained CD-DNNs were integrated with other necessary components of an ASR system to yield functioning ASR systems, which were then evaluated by performing decoding tasks on test data and comparing generated hypotheses with reference transcripts to compute the WERs. The main aim of this work is to compare the two different approaches for cluster tree building, thus apart from the CD acoustic models, the ASR systems are identical.

In order to investigate the influence of senone number on CD AMs' performance, we have built seven cluster trees of different sizes for the DNN based approach and baselines respectively. Table 6.1 lists the test results of these models.

The second line of the table presents the WERs of the ASR systems using DNN based cluster trees. Some previous work showed the recognition accuracy of ASR systems could be improved by increasing the senone number, however after the number of senones reached about 6k, the performance worsened with increasing number of senones. This is because more senones will result in more parameters and thus provide more accurate models, however an excessive number of senones may also lead to some senones' lacking sufficient training data for a robust estimation of their parameters. Our experiments started with cluster trees with 3k senones, which achieved a WER of 19.5%. After this we increased the number of senones by 3k each time, and the WERs on the test dataset continued to decrease until the number of senones reached 18k with a WER value of 18.3%, after which the value began to rise and thus we didn't continue to carry out experiments on DNN based trees with more senones since the performance was unlikely to improve.

The third line lists the test results of baseline systems. Similar to the DNN based approach, WERs of baseline systems kept falling until the number of senones reached 18k, where the WER was 18.4% and above this the WERs tended to rise. Note that 18k was the best case for both DNN based approach and the baseline. The test results of DNN based tree and the baselines can be vividly represented by two parabolas that open upward, as shown in figure 6.1.

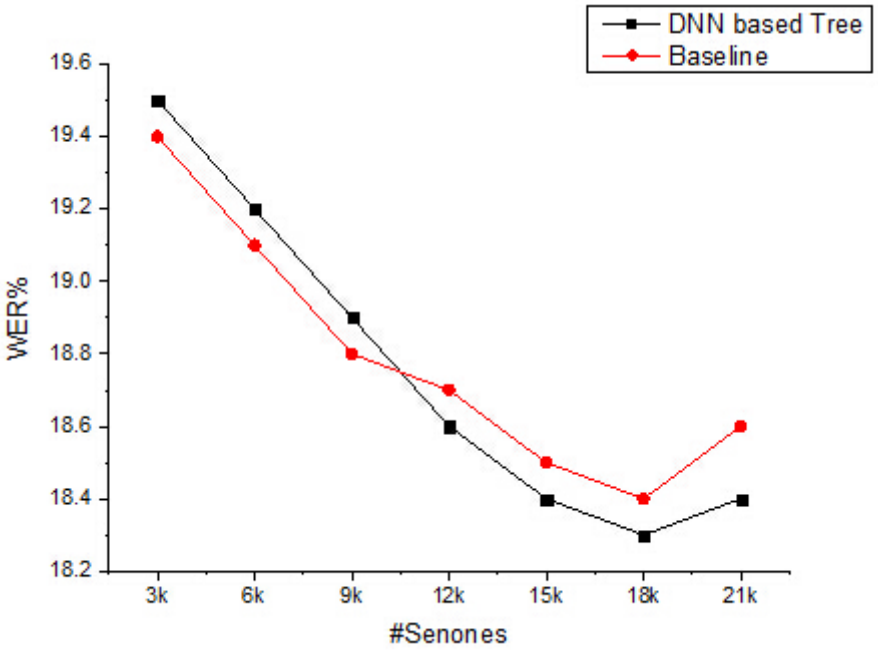

Figure 6.1: *Test results of implemented ASR systems. Both the DNN based approach and the GMM based approach achieved lowest WERs with 18k senones; the DNN based tree with 18k leaves had a WER 18.3% which was lower than the WER of the GMM based tree.*

6.4 Summary

The main purpose of this work is to investigate the performance of our novel approach for cluster tree building, as a comparison the baseline implemented the conventional GMM-based approach which delivers the state-of-the-art recognition accuracy. Figure 6.1 gives a direct impression of the comparison between these two approaches, for the first three systems, namely 3k, 6k and 9k, GMM based cluster trees achieved lower WERs than DNN based trees, or more precisely the WER of each baseline was 0.1$ lower than DNN based trees with the same number of senones. However the WERs of DNN based trees decreased more significantly as the number of senones increased. When the number of senones reached 12k, the performance of the DNN based tree began to exceed the baseline, and for the next four systems DNN based trees outperformed the baseline systems. This is probably because the DNN based approach has a larger advantage from larger numbers of senones. Both approaches attained their optimum performance at 18k senones, and it's worth mentioning that the ASR system using DNN based cluster tree with the 18k senones is best the single system that has the lowest WER 18.3% on the test set dev2012.

6.5 Discussion

In this work we found that CD acoustic models with 18k tied states achieved the best performance for both DNN and GMM based approaches. However, according to our experiments performed two years ago, 6k was the most suitable number of senones, which is much smaller than 18k. The most important reason for this is probably that earlier experiments employed GMM-HMM as the CD acoustic model instead of the DNN-HMM hybrid model, and DNN based systems may have a greater advantage with a large number of senones. Another reason could be that alignments used in earlier experiments were different from those alignments used in this work, as alignments can directly influence the

performance of the systems by providing the feature vectors for the estimation of GMM parameters and also the targets for supervised training of DNN.

7. Conclusion

In this work we have proposed a novel approach to building cluster trees, this novel approach is GMM-free and based on the classification ability of a context independent DNN. Our DNN-based approach utilizes the average CI-DNN outputs for entropy distance measuring instead of the mixture weights as in the conventional GMM-based approach. We have performed multiple experiments to confirm the functionality of our novel approach and allow contrasts to be observed between the novel and conventional approaches. The test results showed that DNN based trees outperform GMM based trees when the number of leaves is relatively large, namely more than 9k.

Another purpose of this work is to investigate the influence of the number of senones on the performance of context dependent acoustic models. Experiments have demonstrated that for both DNN based trees and GMM based trees, WERs of acoustic models decreased as the number of senones increased until the senone number reached 18k, and that after this WERs tended to rise, a function that can be represented by a V-shaped parabola. We began with experiments on 3k senones and for both approaches 18k proved the optimum value. The relative WER reduction for DNN based trees and GMM based trees are 6.1% and 5.1% respectively, which shows that the number of senones can notably influence the performance of acoustic models.

7.1 Further Work

We have shown that DNN based trees have removed the dependence of acoustic modeling on GMMs, however the training of CI-DNN still relies on alignments which can be traced back to a GMM-HMM system. One way in which it is possible to construct an ASR system which is independent of the GMMs is to train the CI-DNN using a flat start and then build the CD acoustic model from alignments provided by the aforementioned CI-DNN.

Further work investigating CD-DNNs with different architectures and training schemes can also be undertaken. Since we exploit the CD-DNNs with the same structure of hidden layers for all numbers of senones, however for a larger output layer more hidden layers can probably provide greater modeling ability. Therefore recognition accuracy can probably be further improved by a CD-DNN with larger architecture or trained with different schemes.

In this work we employed a CD-DNN-HMM hybrid system as the context dependent acoustic model, and experiments have shown that DNN based cluster trees are better suited to a CD-DNN-HMM system than GMM based cluster tree, thus it may be also interesting to consider CD-GMM-HMM systems using DNN based cluster trees.

Bibliography

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines*," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[B+06] C. M. Bishop *et al.*, *Pattern recognition and machine learning*. springer New York, 2006, vol. 4, no. 4.

[BBB+10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.

[BdG+91] L. R. Bahl, P. deSouza, P. Gopalakrishnan, D. Nahamoo, and M. Picheny, "Decision trees for phonological rules in continuous speech," in *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*. IEEE, 1991, pp. 185–188.

[BL+07] Y. Bengio, Y. LeCun *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, 2007.

[BLP+07] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[BLP+12] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[CB10] T. Crowley and C. Bowern, *An introduction to historical linguistics*. Oxford University Press, 2010.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[DYDA12] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[EBC+10] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[FGH+97] M. Finke, P. Geutner, H. Hild, T. Kemp, K. Ries, and M. Westphal, "The karlsruhe-verbmobil speech recognition engine," in *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, vol. 1. IEEE, 1997, pp. 83–86.

[HDY+12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[HES00] H. Hermansky, D. P. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional hmm systems," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 3. IEEE, 2000, pp. 1635–1638.

[HJ89] X. D. Huang and M. A. Jack, "Semi-continuous hidden markov models for speech signals," *Computer Speech & Language*, vol. 3, no. 3, pp. 239–251, 1989.

[HOT06] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[HSW89] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[KHM+14] K. Kilgour, M. Heck, M. Müller, M. Sperber, S. Stücker, and A. Waibel, "The 2014 kit IWSLT speech-to-text systems for English, German and Italian," in *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, 2014, pp. 73–79.

[KN95] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 1. IEEE, 1995, pp. 181–184.

[Low76] B. T. Lowerre, "The harpy speech recognition system," 1976.

[MB90] N. Morgan and H. Bourlard, "Continuous speech recognition using multilayer perceptrons with hidden markov models," in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*. IEEE, 1990, pp. 413–416.

[MHN13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013.

[Mit97] T. M. Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.

[Moh04] M. Mohri, "Weighted finite-state transducer algorithms. an overview," in *Formal Languages and Applications*. Springer, 2004, pp. 551–563.

[Qui86] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, 1988.

[RJ86] L. Rabiner and B.-H. Juang, "An introduction to hidden markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4–16, 1986.

[Run07] A. R. Runnalls, "Kullback-leibler approach to gaussian mixture reduction," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 43, no. 3, pp. 989–999, 2007.

[Sha01] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[SHBL14] A. Senior, G. Heigold, M. Bacchiani, and H. Liao, "Gmm-free dnn acoustic model training," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on.* IEEE, 2014, pp. 5602–5606.

[SKK+07] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio, "A quantitative theory of immediate visual recognition," *Progress in brain research*, vol. 165, pp. 33–56, 2007.

[SMFW01] H. Soltau, F. Metze, C. Fugen, and A. Waibel, "A one-pass decoder based on polymorphic linguistic context assignment," in *Automatic Speech Recognition and Understanding, 2001. ASRU'01. IEEE Workshop on.* IEEE, 2001, pp. 214–217.

[Stu09] S. Stueker, "Acoustic modelling for under-resourced languages," 2009.

[Ver04] K. Vertanen, "An overview of discriminative training for speech recognition," *University of Cambridge*, 2004.

[Vit67] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260–269, 1967.

[VLBM08] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning.* ACM, 2008, pp. 1096–1103.

[YD14] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach.* Springer Publishing Company, Incorporated, 2014.

[YOW94] S. J. Young, J. J. Odell, and P. C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," in *Proceedings of the workshop on Human Language Technology.* Association for Computational Linguistics, 1994, pp. 307–312.

[ZRM+13] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, "On rectified linear units for speech processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 3517–3521.

[ZW14] C. Zhang and P. Woodland, "Standalone training of context-dependent deep neural network acoustic models," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on.* IEEE, 2014, pp. 5597–5601.

List of Figures

List of Tables